MASTER EN SCIENCES INFORMATIQUES

Améliorations de la réalisation du niveau trame d'une liaison X25 et procédures de tests

Stevenne, José

Award date:
1980

Awarding institution:
Universite de Namur

Améliorations de la réalisation du Niveau trame d'une liaison X25 et Procédures de tests

José STEVENNE

Mémoire présenté
en vue de l'obtention du grade
de Licencié et Maître en Informatique

Année académique 1979-1980

Je tiens à remercier les différentes personnes qui ont collaboré à la réalisation de ce travail.

Monsieur Van Bastelaer qui, par ses précieux conseils, m'a permis de mener à bien ce projet.

Monsieur Milgrom qui a mis à ma disposition tout le matériel nécessaire au développement et à la mise au point des programmes.

Monsieur Brunin qui a permis le démarrage du projet de réalisation d'une liaison X25.

Monsieur Lobelle qui, par une constante collaboration, m'a permis de disposer du matériel adéquat tout au long de cette étude.

Messieurs Beudin et Zone qui ont consacré leur mémoire à la réalisation de ce matériel et à l'étude du niveau physique du protocole X25.

Monsieur Art qui a consacré son mémoire à la présentation générale d'une liaison X25 et à la réalisation de son niveau trame.

J. STEVENNE

TABLE DES MATIERES

ANNEXES

Annexe A : Programmation du HDLC

Annexe B : Programmation du DMA

Annexe C : Version de test du programme
du sous-niveau trame 1

Annexe D : Programmes de tests pour
le sous-niveau trame 1

Annexe E : Programmes de l'intégration

- programmes modifiés de SIMTRA
- version de trame 1

BIBLIOGRAPHIE

1) Manuel utilisateur TRANSPAC

2) Microprocesseur
Motorola 6800
Manuel de programmation

3) Microcomputer
Motorola 6800
System design data

4) Conception d'une liaison X25 : lère partie
Présentation générale
Analyse et réalisation du niveau trame
Mémoire de J. Art

5) Réalisation d'une liaison X25 à l'aide d'un microprocesseur
Mémoire de Y. Beudin et G. Zone

6) The Art of Software Testing
Myers

7) Program Test Methods
Hetzel

8) Program Style, Design, Efficiency, Debugging, and Testing
Van Tassel

Chapitre 1 - Introduction

1.1. Avant-propos

Le mémoire qui suit s'intègre dans un projet de réalisation d'une liaison X.25.

Ce projet qui a vu le jour l'année passée a été entamé par quatre étudiants : Messieurs Zone et Beudin de Louvain-la-Neuve en ce qui concerne la partie hardware; Messieurs Art et Lambion pour la partie software.

Le protocole X.25 comprend trois niveaux :

- le niveau physique,
- le niveau trame décomposé en deux sous-niveaux trame 1 et trame 2,
- le niveau paquet.

Le travail avait été réparti comme suit :

- le niveau paquet était traité par Monsieur Lambion,
- le sous-niveau trame 2 était pris en charge par Monsieur Art,
- le sous-niveau trame 1 et la partie physique étaient réalisés par Messieurs Zone et Beudin.

1.2. But du mémoire

Le présent mémoire poursuit deux buts :

Le premier est d'achever la réalisation du niveau trame. Ceci sera effectué par une réécriture du sous-niveau trame 1 et une intégration des sous-niveaux trame 1 et trame 2.

Le second est, partant des différents tests réalisés pour aboutir à cette intégration, de procéder à une réflexion en ce qui concerne les procédures de tests.

Je me suis efforcé d'écrire ce mémoire d'une manière complémentaire à celle de Monsieur Art, c'est-à-dire sous une approche plus pratique. Il constitue en quelque sorte un outil permettant de recommencer aisément tous les tests du niveau trame.

1.3. Structure du mémoire

Le chapitre 1 est consacré à l'introduction.

Le chapitre 2 décrit la version précédente du niveau trame. Nous y trouverons la description des deux sous-niveaux trame 1 et trame 2. Un inventaire des outils de tests existants nous montrera que le sous-niveau trame 2 est au point tandis que le sous-niveau trame 1 est à réécrire entièrement.

Le chapitre 3 présente la nouvelle version du sous-niveau trame 1. Cette version est écrite sous une structure modulaire et articulée autour d'une boucle d'interruptions exposée au paragraphe 3.3. Les deux types d'interruptions possibles, HDLC et DMA, sont présentés

en détail. Ce chapitre expose également les interfaces avec le sous-niveau trame 2 : tant en réception qu'en émission, il décrit les différents buffers et variables communs aux deux sous-niveaux ainsi que leur utilisation.

Le chapitre 4 traite des procédures de tests. Ces tests peuvent être de deux sortes : de conception ou de fonctionnement. Une large partie de ce chapitre est consacrée à l'exposé des tests de conception du niveau trame. Les différents tests effectués pour mettre au point ce niveau y sont présentés dans leur ordre chronologique : le sous-niveau trame 2 d'abord, le sous-niveau trame 1 ensuite, leur intégration enfin. Les tests de fonctionnement seront également pris en compte, mais plus brièvement.

Le chapitre 5 est plus pratique puisqu'il donne la méthode d'utilisation du niveau trame. On y trouvera une description des microprocesseurs et de leur emploi ainsi qu'une présentation de la façon de mettre en oeuvre le niveau trame.

Le chapitre 6 tire les conclusions du travail.

Chapitre 2 - Version précédente du niveau trame : description, critique, améliorations envisagées.

2.1. L'avis X.25

Afin de définir les différents éléments qu'il comprend, il est utile de rappeler ici la définition de l'avis X.25. (*)

L'avis X.25 est intitulé comme suit :
"Interface entre équipement terminal de traitement de données (ETTD) et équipement de terminaison du circuit de données (ETCD) pour terminaux fonctionnant en mode paquet, raccordés à un réseau public de transmission de données".

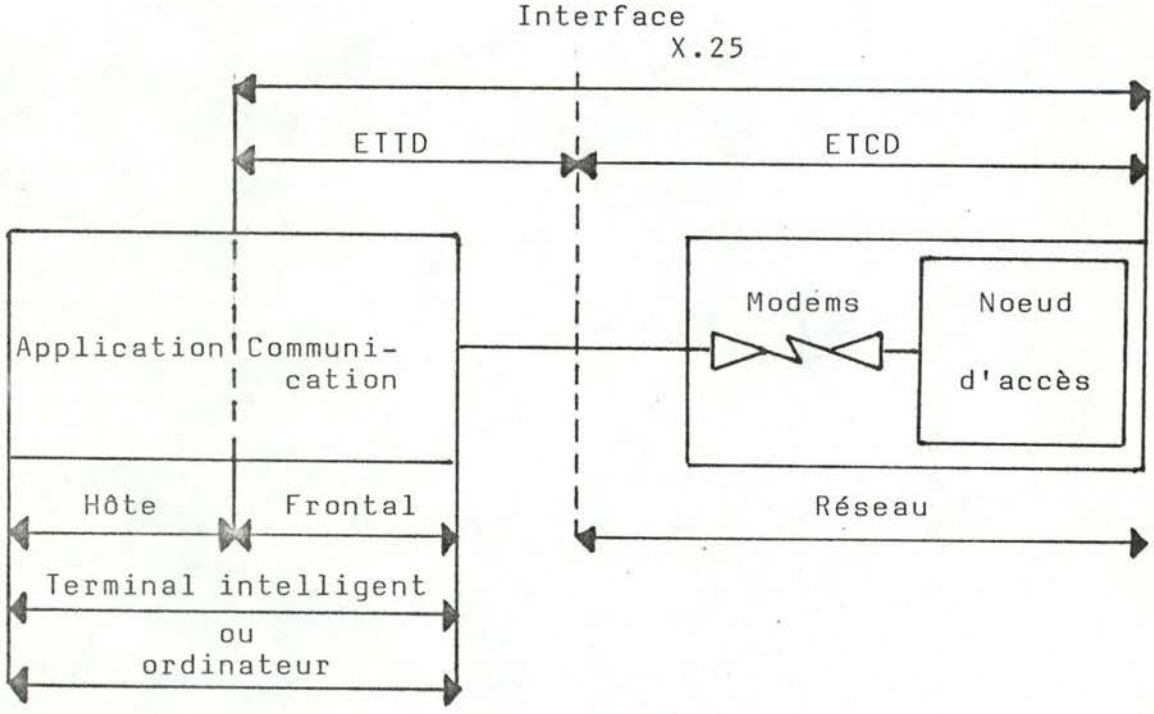


Fig. 2.1 - Jonction X.25

(*) Cette définition a été reprise du Mémoire de Monsieur Art.

L'ETTD est l'équipement de l'abonné qui produit les paquets, les envoie vers le réseau, reçoit les paquets provenant du réseau et en extrait l'information. Il s'agit physiquement d'un ordinateur hôte, d'un processus frontal ou d'un terminal intelligent. l'ETCD est l'équipement du réseau auquel est connecté l'ETTD. X.25 définit donc l'interface local entre l'utilisateur et le réseau.

2.2. Découpage du niveau trame en sous-niveaux

Pour une description détaillée du fonctionnement du niveau trame, le lecteur pourra consulter le mémoire de Monsieur Art. Nous rappellerons seulement ses grands principes.

Le niveau trame sert, en émission, à transporter sur la liaison ETTD-ETCD les paquets construits par le niveau paquet et, en réception, à passer au niveau paquet ceux qu'il a récoltés sur la ligne.

Il assure la détection des erreurs de transmission, leur correction par retransmission des trames erronées, la transparence ainsi que les procédures d'établissement et de libération du lien.

Comme le montre la figure 2.2, il est décomposé en deux sous-niveaux. Le sous-niveau trame 1 transfert les trames de trame 2 vers la ligne pour l'émission et de la ligne vers trame 2 pour la réception. Il détecte aussi les erreurs de transmission et élimine les trames fautives. Le sous-niveau trame 2 réalise la gestion effective des trames : il les analyse, contrôle leur séquence et les passe du niveau paquet vers trame 1 en émission et de trame 1 vers le niveau paquet en réception.

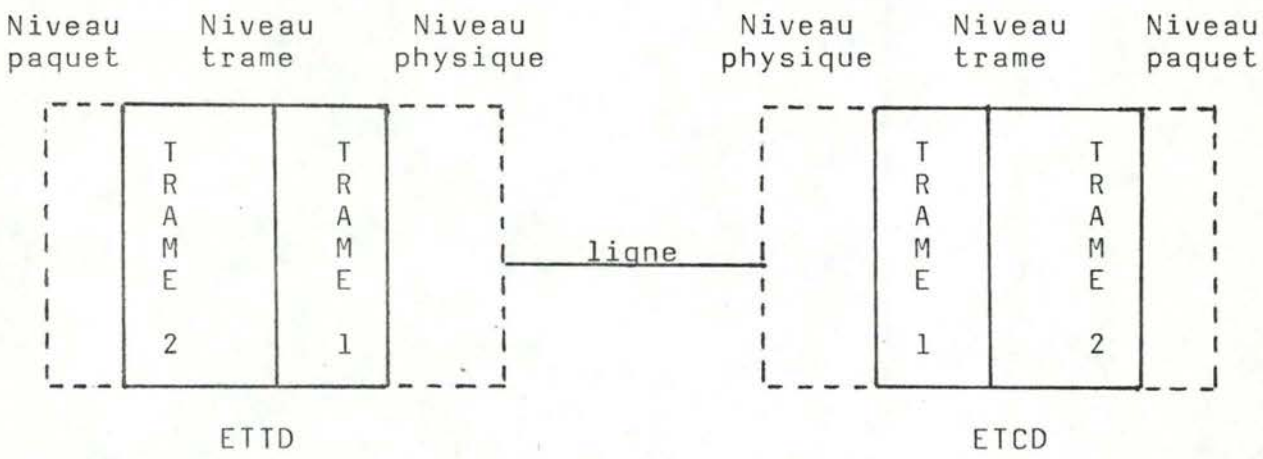


Fig. 2.2 - Les sous-niveaux du niveau trame

2.3. Le sous-niveau trame 1

Le sous-niveau trame 1 réalise les deux fonctions suivantes : il gère la ligne en émission et en réception (voir fig. 2.3).

a) Gestion de la ligne en émission

- prise en charge des trames à émettre,
- calcul du FCS,
- insertion de "o" pour la transparence,
- ajoute de fanions pour la synchronisation,
- avortement des trames inutiles,
- envoi de bits en série sur la ligne,
- remplissage entre les trames.

b) Gestion de la ligne en réception

- détection des bits en série,
- suppression des fanions de synchronisation,
- suppression des "o" de transparence,
- contrôle du FCS,

- élimination des trames invalides,
- passage des champs adresse, commande et information des trames correctes au sous-niveau trame 2.

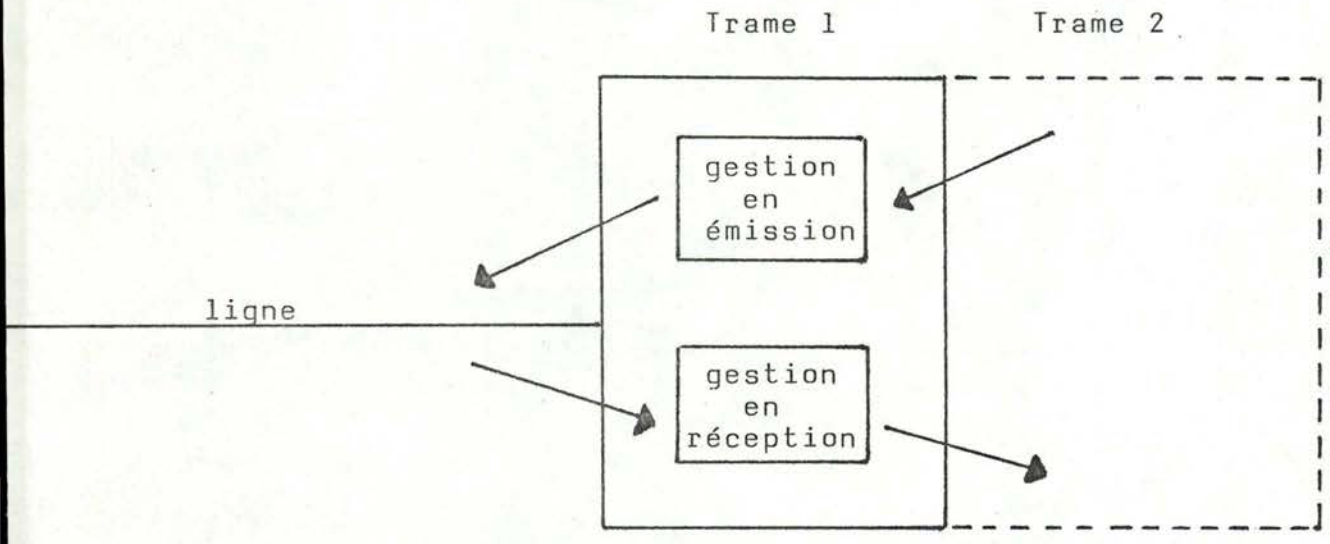


Fig. 2.3 - Fonctions du sous-niveau trame 1

Ce sous-niveau avait été écrit en assembleur par Messieurs Zone et Beudin.

Cette version n'ayant jamais donné de résultats concrets, la première idée fut de la corriger pour ensuite l'intégrer avec le sous-niveau trame 2.

Dès la lecture du programme, ce projet s'est avéré trop difficile à mettre en oeuvre.

D'une part, le programme était mal documenté. Certains passages étaient même complètement illisibles car dépourvus de commentaires.

D'autre part, si les spécifications étaient correctes et ne demandaient donc aucun changement, la structure du travail ne semblait pas très rigoureuse. Il en résultait une difficulté presque insurmontable pour la réalisation des tests à effectuer.

L'idée de la correction du programme fut donc abandonnée et sa réécriture s'imposait avec un double objectif.

Le premier était l'inclusion d'une documentation nécessaire et suffisante pour une lecture aisée. Un nombre important de commentaires est indispensable car le programme est écrit en assembleur, c'est-à-dire dans un langage de bas niveau.

Il fut aussi décidé d'y ajouter une version équivalente dans un langage de haut niveau qui serait ici le langage C.

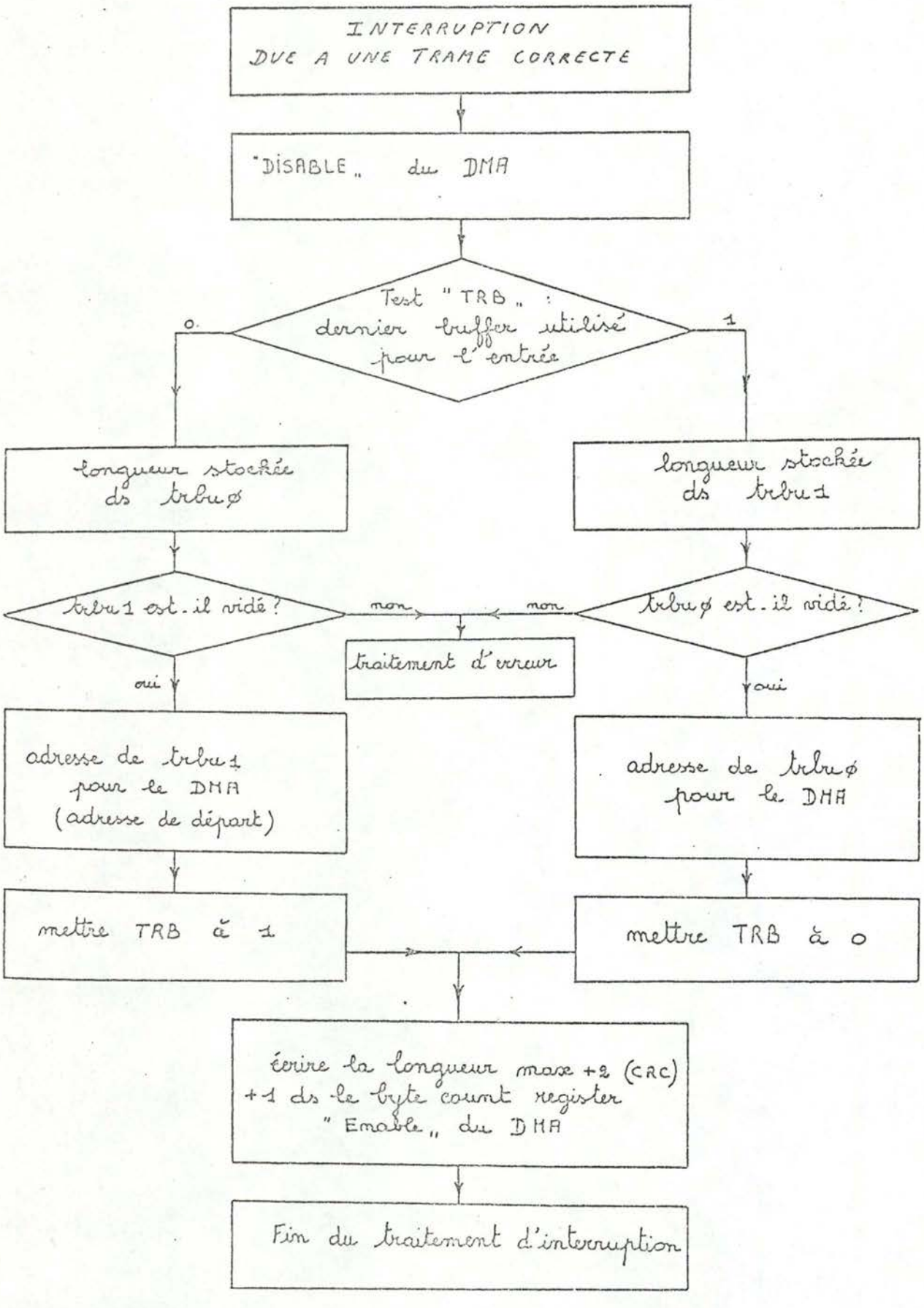
Le second but résidait dans la forme du programme. Il sera réécrit sous une structure plus modulaire par une série d'appels à des sous-routines.

Si cette idée présentait un inconvénient du côté de la performance, elle procurait aussi un double avantage : une structuration plus prononcée accroît la facilité dans le travail des tests; elle augmente également la lisibilité du programme si le degré de modularité reste convenable.

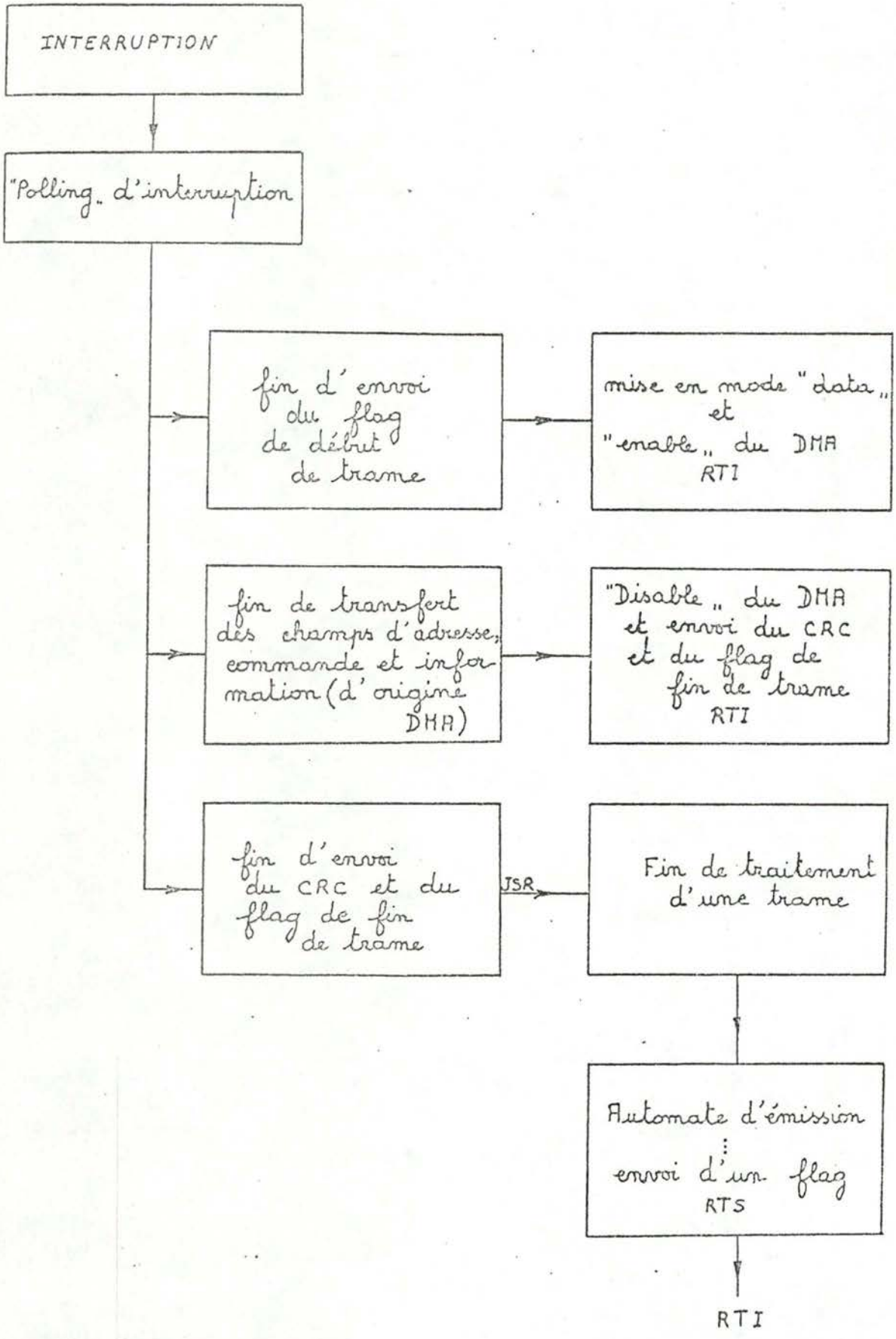
La version précédente du sous-niveau trame 1 présentait trois modules dont les ordinogrammes se trouvent aux figures 2.4, 2.5 et 2.6.

Le module RT effectue les opérations à faire lors d'une réception de trame : gestion des buffers et calcul de la longueur.

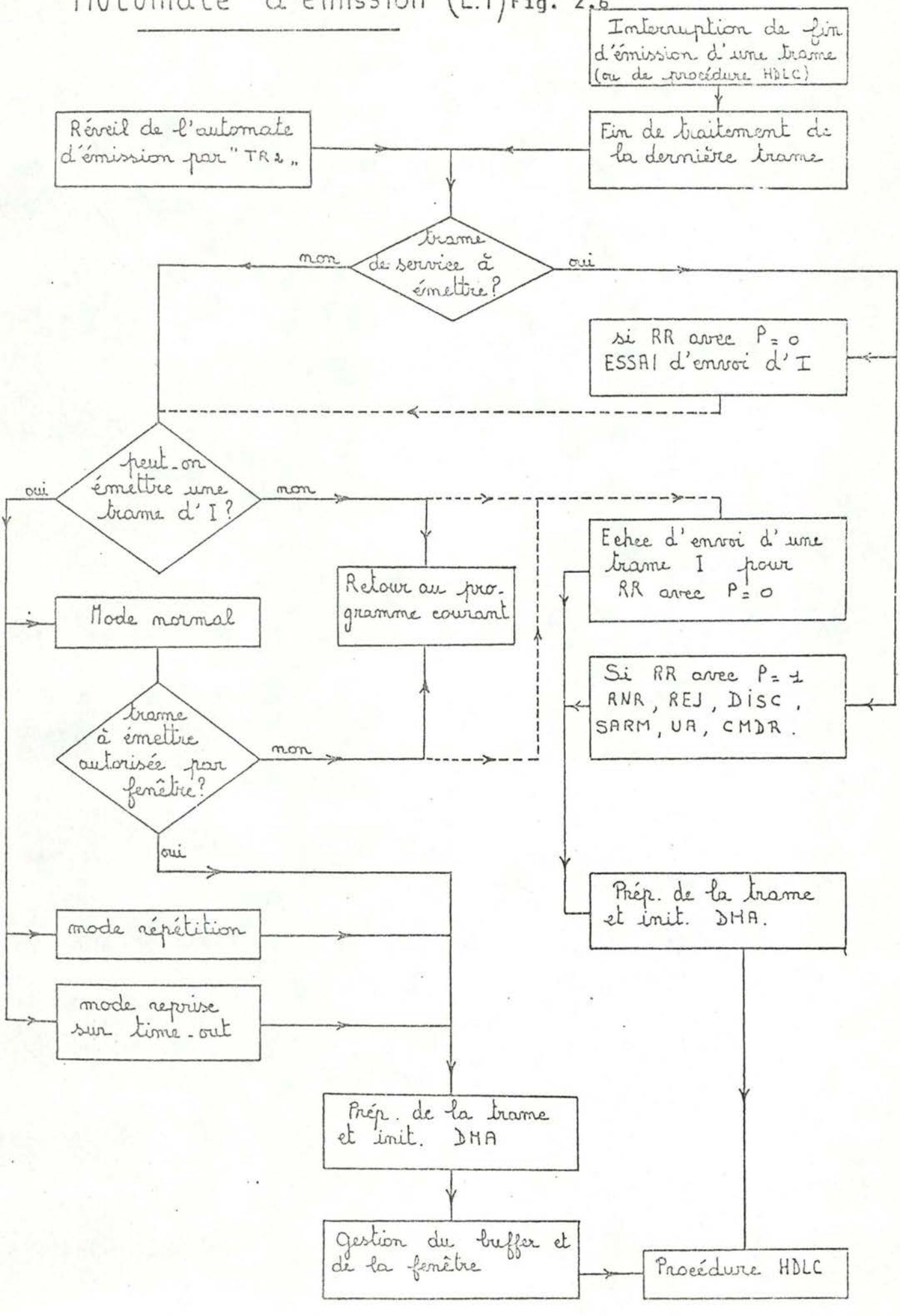
Automate de réception des trames (R.T.) Fig. 2.4



PROCEDURE HDLC Fig. 2.5



Automate d'émission (E.T) Fig. 2.6



La procédure HDLC réalise l'envoi de la trame proprement dite; cette trame se trouve dans un buffer dmaout et doit partir sur la ligne. Les exécutions suivantes sont réalisées : envoi d'un flag, envoi de la trame par DMA et envoi du FCS.

Le dernier module ET est le plus complexe. Il prépare dans le buffer dmaout les différents champs de la trame à envoyer par la procédure HDLC.

2.4. Le sous-niveau trame 2

Le sous-niveau trame 2 réalise comme première fonction l'analyse des trames correctement reçues par trame 1, le contrôle de l'adresse et la séparation des commandes et des réponses.

De plus, le primaire prend en charge les paquets à émettre, génère les trames de commandes, gère le temporisateur et traite les réponses.

Enfin, le secondaire traite les commandes, génère les réponses et extrait les paquets des trames d'information reçues et les passe au niveau paquet.

Les programmes concernant ce sous-niveau ont été écrits par Monsieur Art. Ils ont été testés puis regroupés pour former SIMTRA. Cette configuration, qui a permis de tester entièrement le sous-niveau trame 2, est décrite au paragraphe 2.5.

On peut considérer que ce sous-niveau est totalement terminé et fonctionne correctement.

Mais, comme pour le sous-niveau trame 1, il nous a semblé que les programmes manquaient de commentaires. Nous nous sommes donc efforcés de les documenter avec un maximum de précision.

2.5. Outils de tests existants

Comme nous l'avons signalé au paragraphe 2.3, il n'existe aucun outil de test pour le sous-niveau trame 1.

En ce qui concerne le sous-niveau trame 2, l'outil essentiel pour tester son fonctionnement est SIMTRA. Sa configuration est présentée à la figure 2.7.

Nous n'allons pas décrire ici les procédures de tests qui ont permis d'aboutir à SIMTRA. Ce sujet est traité au paragraphe 4.2.1. Nous nous contenterons de décrire l'outil ainsi que, sommairement, son fonctionnement.

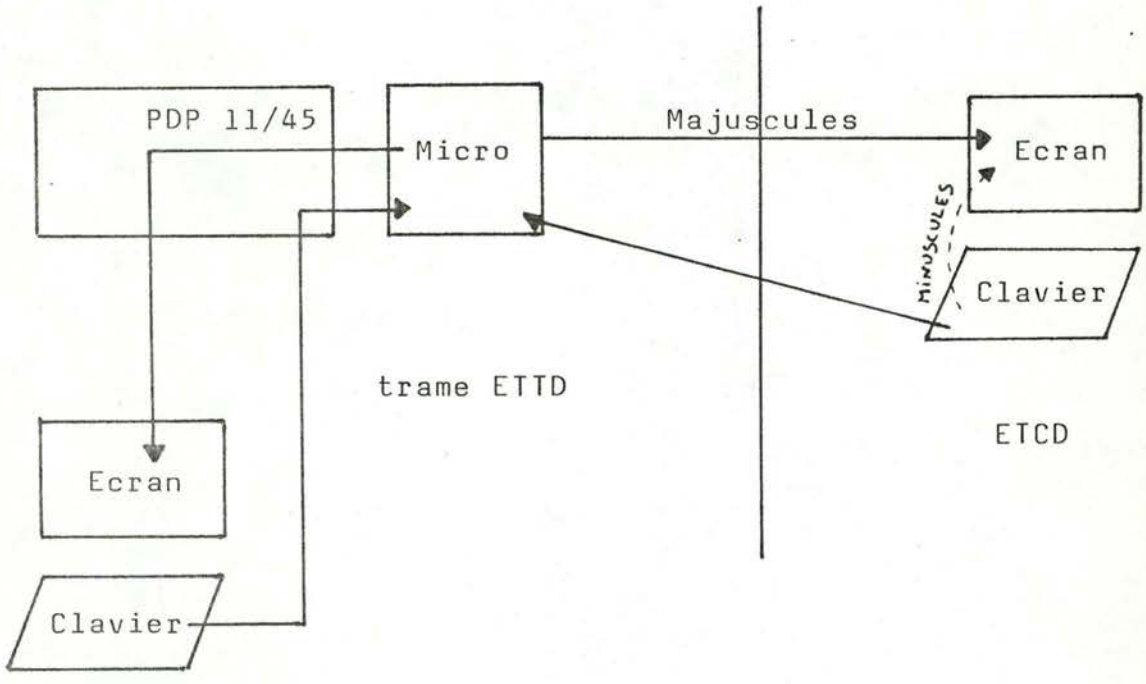


Fig. 2.7

L'ETTD est constitué de deux parties : le sous-niveau trame 2 du niveau trame se trouve dans un micro-processeur. Il est relié au niveau paquet qui est dans la PDP.

L'ETCD ne comprend qu'un écran et un clavier.

Un opérateur joue le rôle de l'ETCD. Il converse avec l'ETTD au moyen de son terminal. Il peut envoyer des trames à l'ETTD au moyen du clavier et en recevoir sur l'écran.

Du côté de l'ETTD, l'autre opérateur peut envoyer des paquets et des messages de la PDP vers le micro-processeur, grâce à un programme tournant sur la PDP.

Cette configuration a permis d'effectuer les tests complets des fonctions de l'automate en vérifiant, à partir des états possibles de l'automate, si ses transitions et ses actions sont correctes pour chaque événement.

De plus, SIMTRA a également permis de tester l'interface trame-paquet.

Cet outil étant tout à fait correct, aucune modification, excepté l'intégration de trame 1, n'y a été apportée.

Chapitre 3 - Nouvelle version du sous-niveau trame 1.

3.1. Spécifications du sous-niveau trame 1

3.1.1. Spécifications fonctionnelles

Comme nous l'avons déjà signalé au paragraphe 2.2, le sous-niveau trame 1 assure la gestion de la ligne en émission et en réception.

Nous n'expliciterons pas entièrement ici les fonctions de trame 1 mais nous les citerons en faisant référence aux différents paragraphes qui les décrivent.

Avant d'émettre ou de recevoir des trames, le sous-niveau trame 1 effectue une initialisation qui est la phase d'établissement de la communication. Cette phase qui comprend l'échange de signaux d'appel est explicitée au paragraphe 3.2.

Pour gérer la ligne en réception, le sous-niveau trame 1 récolte les trames arrivant de la ligne et, les passe au sous-niveau trame 2 après avoir effectué un contrôle d'erreur grâce au FCS. Le point 3.3 détaille cette procédure en décrivant notamment l'interface entre trame 1 et trame 2 du côté de la réception.

En ce qui concerne la gestion de la ligne en émission, le rôle du sous-niveau trame 1 est plus important. Il est entièrement repris au paragraphe 3.4 mais nous allons néanmoins citer ses trois fonctions essentielles.

Tout d'abord, il prend les trames analysées et passées par le sous-niveau trame 2, construit et y insère des champs d'adresse et de commande et envoie le tout sur la ligne, l'HDLC réalisant le bit stuffing et le calcul du FCS.

Ensuite, il assure la mise à jour de la fenêtre et de pointeurs utilisés par trame 2.

Enfin, il effectue l'émission de flags entre les différentes trames expédiées.

3.1.2. Contraintes de conception

La réalisation du sous-niveau trame 1 s'est opérée à l'aide du HDLC et du DMA (dont on trouvera les documentations en annexe A et B) pour les transferts.

Le HDLC permet de réaliser des fonctions telles que le bit stuffing et le calcul du FCS de façon purement hardware. Cette technique offre une plus grande rapidité et permet de décharger le microprocesseur pendant ces opérations.

Le DMA, quant à lui, permet d'accroître le taux de transmission. De plus, il réalise lui-même les rangements des trames transférées, permettant au microprocesseur d'effectuer d'autres opérations.

L'utilisation de ces deux techniques implique l'emploi de registres d'interruption.

D'une part, en ce qui concerne l'HDLC, le registre IR va nous renseigner sur l'opération terminée :

- fin de réception sans erreur,
- fin de réception avec erreur (overrun, trame invalide,...)

- fin d'émission sans erreur,
- fin d'émission avec erreur (underrun si le THR n'a pas été garni assez rapidement par l'HDLC).

D'autre part, pour le DMA, les registres CCR0 en émission et CCR1 en réception nous permettent de savoir, par l'intermédiaire du registre IC, s'il y a une fin d'opération DMA.

Une remarque peut, dès à présent, être formulée à propos de l'emploi de ces registres d'interruption : il va nous apporter deux avantages.

Il va tout d'abord faciliter le découpage du sous-niveau trame 1 car nous pourrons créer un module par cause d'interruption (voir le paragraphe 3.3).

Il va également nous aider dans l'intégration des deux sous-niveaux trame 1 et trame 2. En effet, SIMTRA étant justement articulé autour d'une boucle d'interruption, il suffira d'inclure dans cette boucle les nouvelles interruptions dues au HDLC et au DMA (voir le paragraphe 4.2.3.2).

Les modules concernant le sous-niveau trame 1 sont en annexe C.

3.2. Initialisations du niveau trame 1

3.2.1. Initialisation du X21 bis et du HDLC

2.2.1.1. Principe du X21 bis

Le protocole X25 spécifie que le niveau bit (fonction physique) doit être conforme à l'avis X21 ou X21 bis du CCITT (*). Messieurs Zone et Beudin optèrent pour le X21 bis car il paraissait à priori plus facilement adaptable.

(*) Comité Consultatif International télégraphique et téléphonique.

Les signaux servant au "handshaking" de la jonction physique sont les suivants :

- les circuits 104, 106, 107, 109, 142 pour l'ETCD,
- les circuits 103, 105, 108/1 pour l'ETTD.

Ces signaux sont repris à la figure 3.1.

Avis V.24 Circuit de jonction n°	Désignation du circuit de jonction
102	Terre de signalisation ou retour commun
103	Emission des données
104	Réception des données
105	Demande pour émettre
106	Prêt à émettre
107	Poste de données prêt
108/1	Connectez le poste de données sur la ligne
109	Détecteur du signal de ligne reçu sur la voie de données
114	Base de temps pour les éléments de signal à l'émission (ETCD)
115	Base de temps pour les éléments de signal à la réception (ETCD)
142	Indicateur d'essai (ETCD)

Fig. 3.1 - Circuits de la jonction X21 bis

L'ETTD demande la connexion du poste de données sur la ligne en fermant le circuit 108/1.

L'ETCD répond que le poste de données est prêt en fermant le circuit 107.

L'ETTD demande pour émettre en fermant le circuit 105.

L'ETCD répond qu'il est prêt à émettre en fermant le circuit 106.

Cet enchaînement peut se schématiser par la figure 3.2.

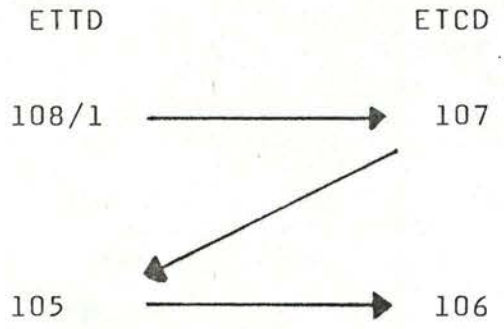


Fig. 3.2 - Initialisation de la jonction X21 bis

3.2.1.2. Principe du HDLC

Pour la gestion de la ligne en émission, le HDLC insère des 0 pour la transparence, ajoute des fanions pour la synchronisation, assure l'envoi des flags, des trames et du FCS.

En réception, il supprime les 0 de transparence, ôte les fanions, passe la trame au DMA et vérifie le FCS.

Le HDLC possède cinq registres dont voici les fonctions essentielles :

a) premier registre de commande (CR1) qui permet d'activer l'HDLC en émission et/ou réception et de sélectionner l'émission désirée (flag, data ou FCS);

b) deuxième registre de commande (CR2) qui permet entre autre de se positionner en mode auto-flag et de sélectionner le nombre de bytes de contrôle (1 ou 2);

c) troisième registre de commande (CR3) qui détermine la longueur du "residual byte";

d) registre d'état qui permet d'obtenir des renseignements sur l'état du HDLC.

e) registre d'interrupt qui permet de savoir quelle opération vient d'être terminée.

Nous rappelons que l'annexe A est consacrée à la description détaillée du HDLC.

Seuls les deux premiers registres de commande sont initialisés au départ puisqu'ils participent à l'établissement de la jonction physique (voir fig. 3.3 et 3.4), les autres étant positionnés aux endroits adéquats du programme.

3.2.1.3. Initialisation du X21 bis et du HDLC côté ETTD

L'ordinogramme de cette initialisation se trouve à la figure 3.3.

On peut l'interpréter de la façon suivante :

Pour que l'ETTD décide de se connecter, le microprocesseur ferme la ligne 108/1 en forçant un 1 sur MISCOUT c'est-à-dire en mettant FE dans le control register n° 1 du HDLC (CR1). Ensuite, il va lire l'état de DSR dans SR jusqu'à ce qu'il soit dans l'état logique 1. Quand DSR est à 1, cela signifie que 107 est fermé ou encore que le poste de données est connecté sur la ligne. Pour émettre, le microprocesseur active le HDLC en transmission-réception mode data pour envoyer des flags. L'activation de cette transmission a pour effet de mettre RTS à 1 c'est-à-dire de fermer 105. Il reste à attendre la réponse de l'ETCD par l'intermédiaire du bit 1 de IR.

3.2.1.4. Initialisation côté ETCD

En ce qui concerne l'ETCD, la démarche, dont l'ordinogramme est à la figure 3.4, est identique.

Au début, l'ETCD ferme le circuit 107 (les électroniciens ont choisi de le fermer dès le départ, donc il n'y a rien à faire). L'ETCD teste ensuite SR pour voir si CD = 1, c'est-à-dire pour tester la demande pour émettre de l'ETTD. Une fois le test positif, l'ETCD se place en réception transmission mode data en mettant 3F dans CR1 et en auto-flag en mettant EF dans CR2. La transmission ainsi activée éveille le signal RTS, ce qui permet l'émission de flags. Il reste à attendre le délai de 40 msec avant de mettre 0 dans le PIA 1 pour signaler qu'on est prêt à émettre (fermeture de 106).

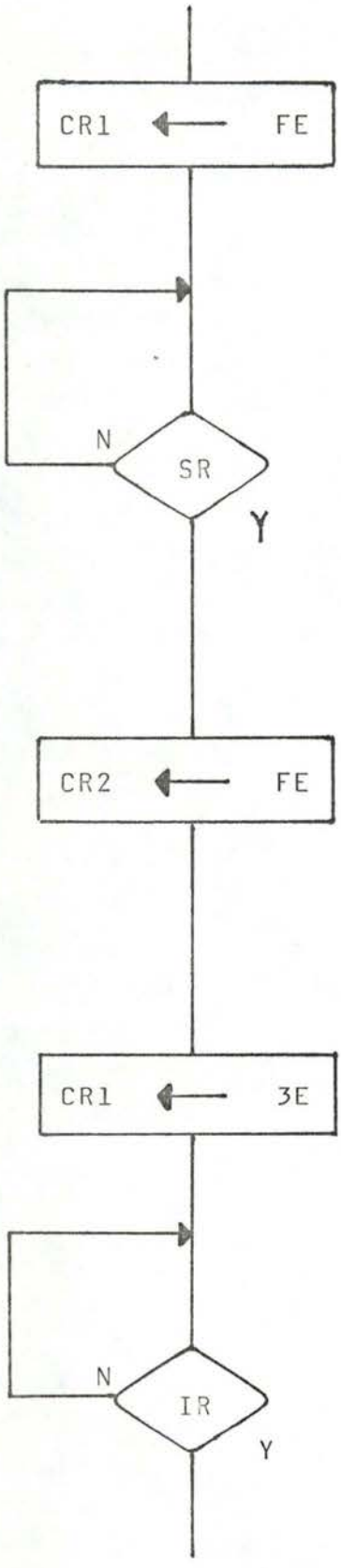


Fig. 3.3 - Initialisation du X21 bis et
du HDLC côté ETTD

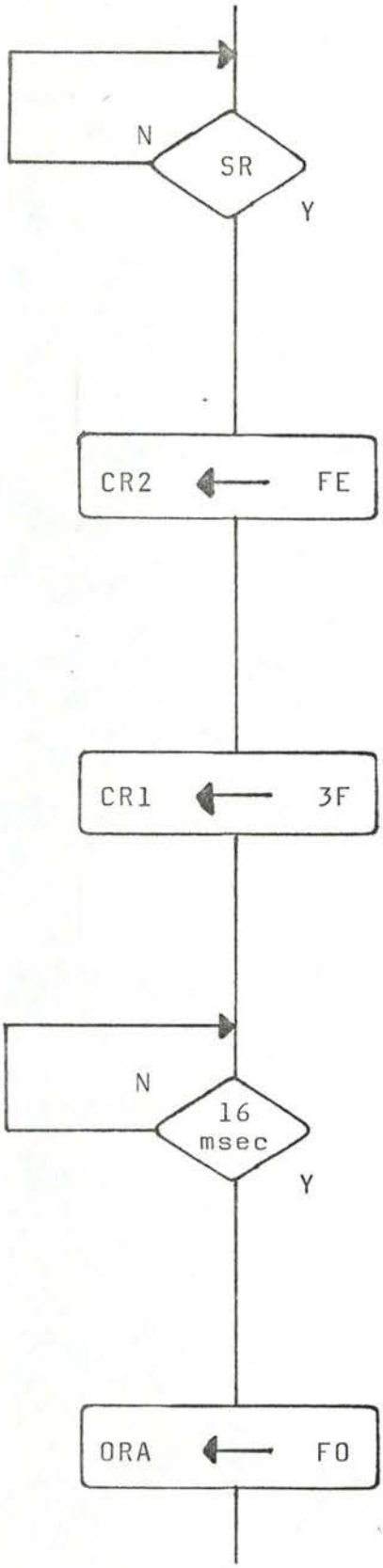


Fig. 3.4 - Initialisation du X21 bis et
du HDLC côté ETCD

3.2.2. Initialisation du DMA

3.2.2.1. Principe du DMA

Le DMA (direct memory access) est un mode de transfert de blocs d'information entre le microprocesseur et ses périphériques. Il permet d'effectuer des échanges d'informations tout en libérant le microprocesseur pour d'autres tâches. Le fonctionnement du microprocesseur n'est donc interrompu que pendant la demande de transfert; le DMA auquel on a fourni les informations nécessaires au contrôle (adresse, longueur) réalisant ce transfert pendant que le microprocesseur effectue d'autres actions.

Le DMA possède quatre canaux : cela signifie qu'il peut recevoir quatre demandes distinctes dont les priorités sont établies par la programmation.

Nous nous contenterons de donner ici la liste des registres utilisés par le DMA et pour chacun d'eux leurs fonctions principales. Pour plus de détails, le lecteur pourra consulter avantageusement l'annexe B.

a) 4 registres de contrôle de canal (1 par canal) qui permettent de savoir si les canaux sont utilisés en lecture ou écriture et s'il y a une fin d'opération DMA sur le canal correspondant au numéro de registre.

b) 1 registre de contrôle qui permet d'accepter les transferts sur les canaux adéquats.

c) 1 registre d'interrupts qui permet d'autoriser les interrupts pour les canaux utilisés.

d) 1 registre de "data chain" qui donne, entre autre, le nombre de canaux.

e) 4 registres adresse et 4 registres de longueur pour les transferts d'informations.

3.2.2.2. Initialisation du DMA

Cette initialisation consiste à garnir les registres afin de présenter la configuration des canaux de transferts, de signaler les interruptions autorisées et de préparer les adresse et compteur nécessaires à l'échange des informations.

a) Registres de contrôle de canal (CCR0, CCR1, CCR2, CCR3) : dans le cas présent, deux canaux seulement sont utilisés; le canal 0 en émission et le canal 1 en réception. On mettra donc 04 dans CCR1 et 05 dans CCR0.

b) Registre de contrôle (PC) : il autorise les transferts en réception ou en émission. Ici, on autorisera initialement la réception en mettant 02 dans le PC.

c) Registre d'interrupts (IC) : on autorise les interrupts à la fin des émissions et des réceptions. L'IC sera initialisé à 03.

d) Registre data chain (DCR) : on y mettra 08 pour signaler qu'il y a quatre canaux.

e) Registres adresses (AR0, AR1, AR2, AR3). Le registre d'adresse de réception AR1 est initialisé au début de trbu 0 car c'est dans ce buffer que l'on recevra la première trame.

f) Les registres compteurs (BCR0, BCR1, BCR2, BCR3) : on initialise le registre compteur de réception BCR1 à la valeur 140 c'est-à-dire la valeur égale à la longueur des buffers de réception (trbu 0 et trbu 1). Cette valeur est supérieure à la longueur maximum d'une trame correcte ce qui permet d'éviter un interrupt DMA à la fin de la réception d'une trame normale. Ceci permettra ainsi de répondre par un CMDR à des trames légèrement trop longues et d'effectuer une procédure d'erreur pour les trames supérieures à 140 bytes.

Une remarque s'impose ici : les initialisations DMA ne sont faites que pour la réception car celle-ci peut arriver à tout moment (c'est un événement externe) tandis que pour l'émission, les registres peuvent être garnis juste avant d'expédier la trame sur la ligne.

3.3. La boucle d'interruption

La structure de la boucle d'interruption est représentée sur le schéma 3.6.

Cette boucle est composée de six types d'interrupts (fig. 3.5).

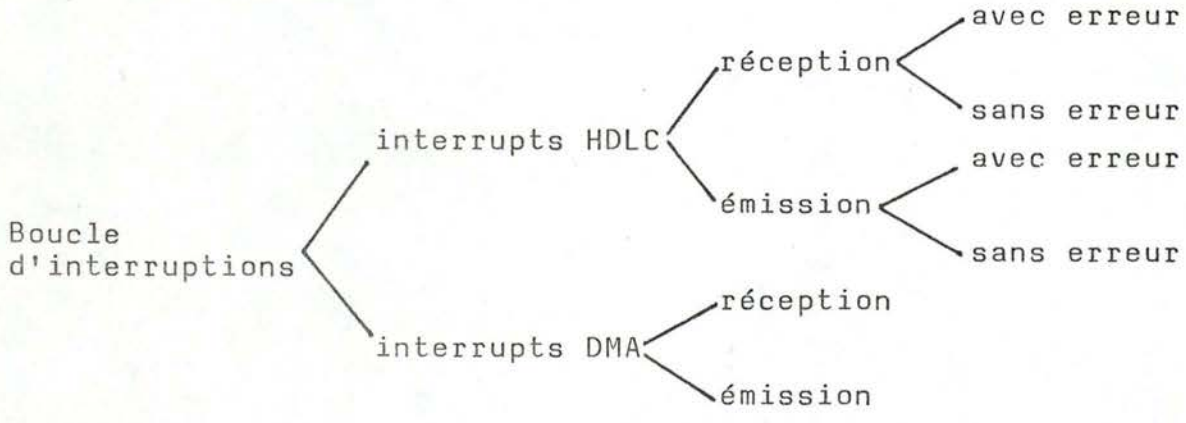


Fig. 3.5 - Les six types d'interrupts

Il a bien fallu effectuer un choix quant à l'ordre de traitement des interruptions.

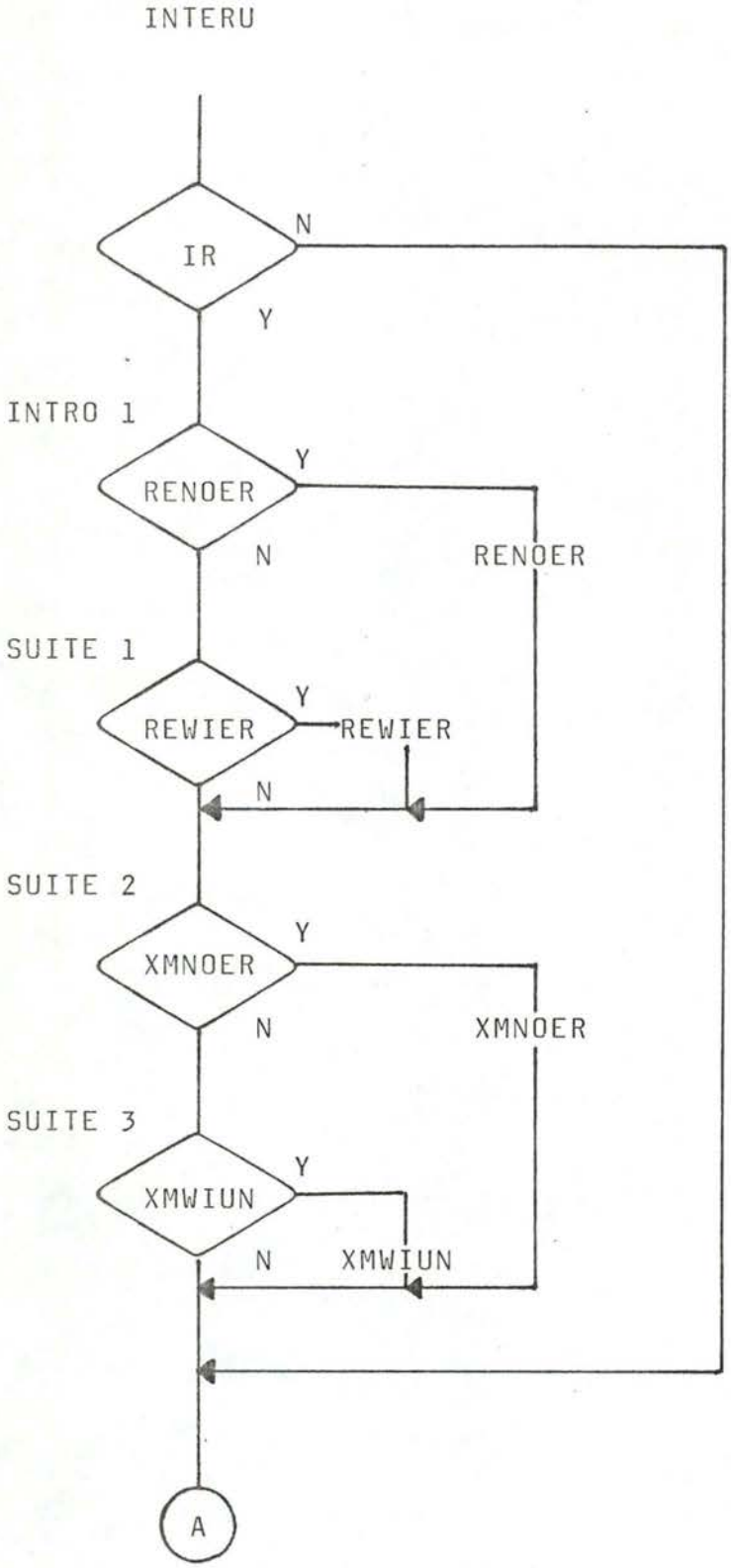
Les interrupts HDLC seront traités en premier lieu car leurs arrivées sont d'une probabilité supérieure à celles des interrupts DMA.

En effet, nous ne pouvons normalement obtenir qu'une seule sorte d'interruption DMA, pour la fin de réception (la fin d'émission ne pouvant théoriquement pas se produire), et encore n'avons nous pour celle-ci qu'une interruption par trame.

Par contre, le HDLC donne des interrupts en émission et en réception. De plus, ces interrupts sont multiples pour une trame : flag, data, FCS.

Nous avons également opté pour donner la priorité aux interruptions dues aux réceptions car elles constituent des événements extérieurs. Si l'interrupt de la réception n'est pas traité rapidement, on pourrait perdre la trame suivante, ce qui impliquerait sa réémission du côté de l'émetteur; tandis que si c'est l'interrupt de l'émission qui n'est pas servi immédiatement, cela entrainera seulement le départ d'un flag en plus sur la ligne.

interruptions
HDLC



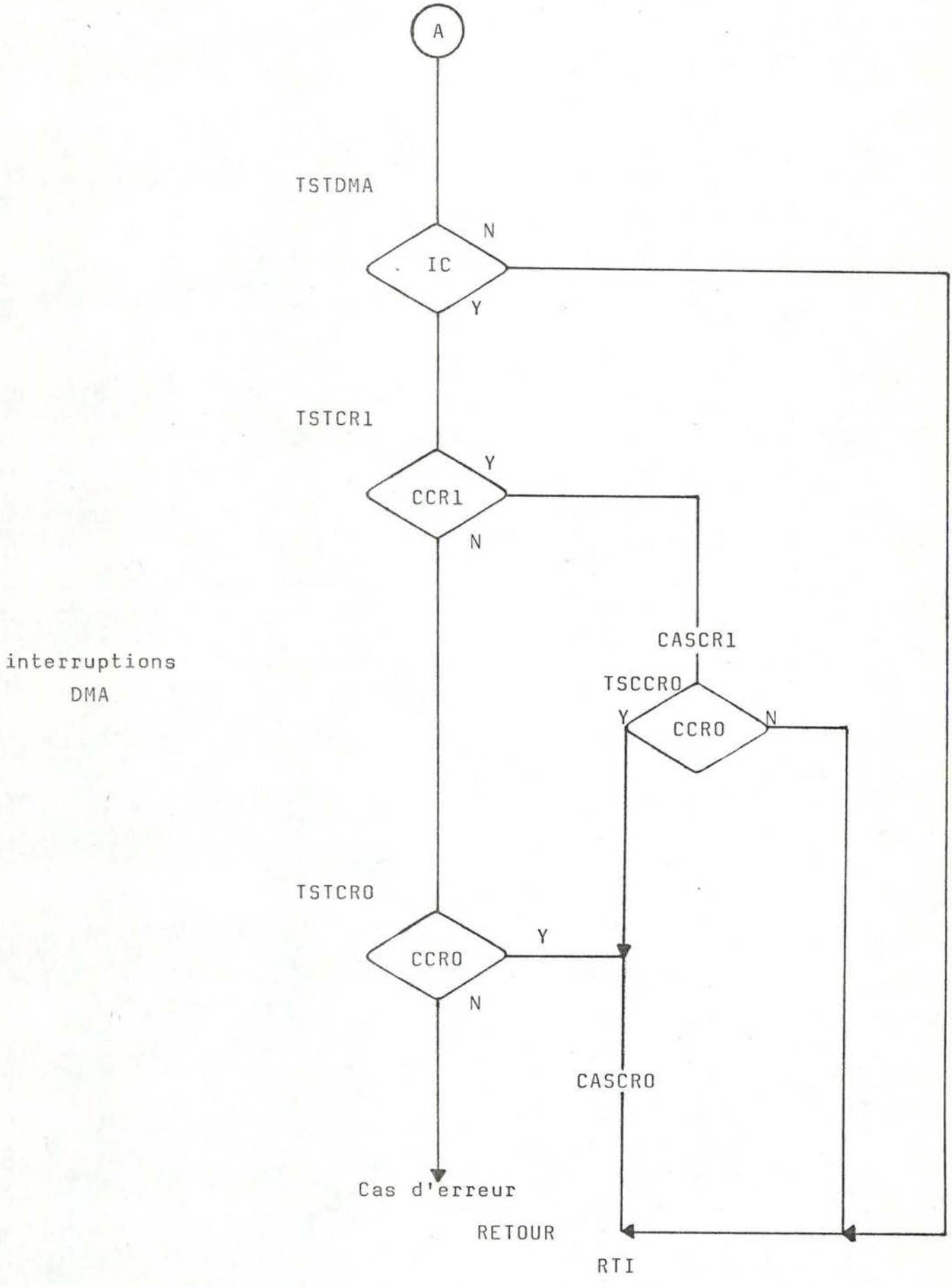


Fig. 3.6 - Structure de la boucle d'interruption

3.4. Structure du sous-niveau trame 1 côté réception

3.4.1. Description de l'interface avec trame 2

Les trames reçues de la ligne sont rangées alternativement dans deux tampons d'entrée. L'un sert à ranger la trame qui est reçue de la ligne, l'autre contient la trame traitée par l'automate de trame 2. Une zone, en début de chaque tampon, est réservée pour ranger la longueur de la trame contenue (voir figure 3.7). Le champ est garni par trame 1 dès que la trame peut être traitée par trame 2. Il est remis à 0 dès que la trame a été traitée, ce qui indique que le tampon est vide et peut donc à nouveau être rempli par trame 1.

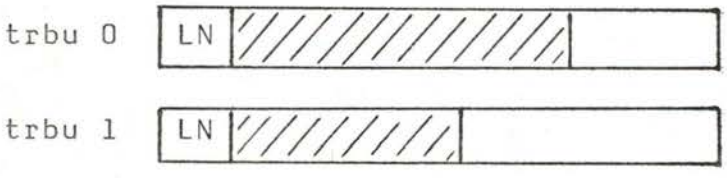


Fig. 3.7 - Buffers de réception

3.4.2. Algorithmes

Comme nous l'avons signalé dans les paragraphes précédents, nous pouvons distinguer trois types d'interruption en réception : soit une fin de réception HDLC sans erreur, soit une fin de réception HDLC avec erreur, soit une fin de réception DMA. Ceci va nous donner trois modules de traitement de réception.

a) Fin de réception HDLC sans erreur

La première opération à réaliser est l'arrêt du DMA c'est-à-dire qu'on remet le bit request enable du PC à 0. Ensuite on calcule la longueur en ôtant BCR1 plus deux (car il faut enlever les 2 bytes du FCS) à la longueur initiale. On range cette longueur en début de zone et on passe à l'autre buffer en modifiant la variable de travail (Switch) TRB et le registre adresse AR1. Il reste alors à tester la longueur de ce buffer. Si elle est nulle; on peut continuer en relançant le DMA c'est-à-dire en repositionnant le bit du PC. Sinon, on effectue le traitement d'erreur.

Cet enchaînement est représenté à la figure 3.8.

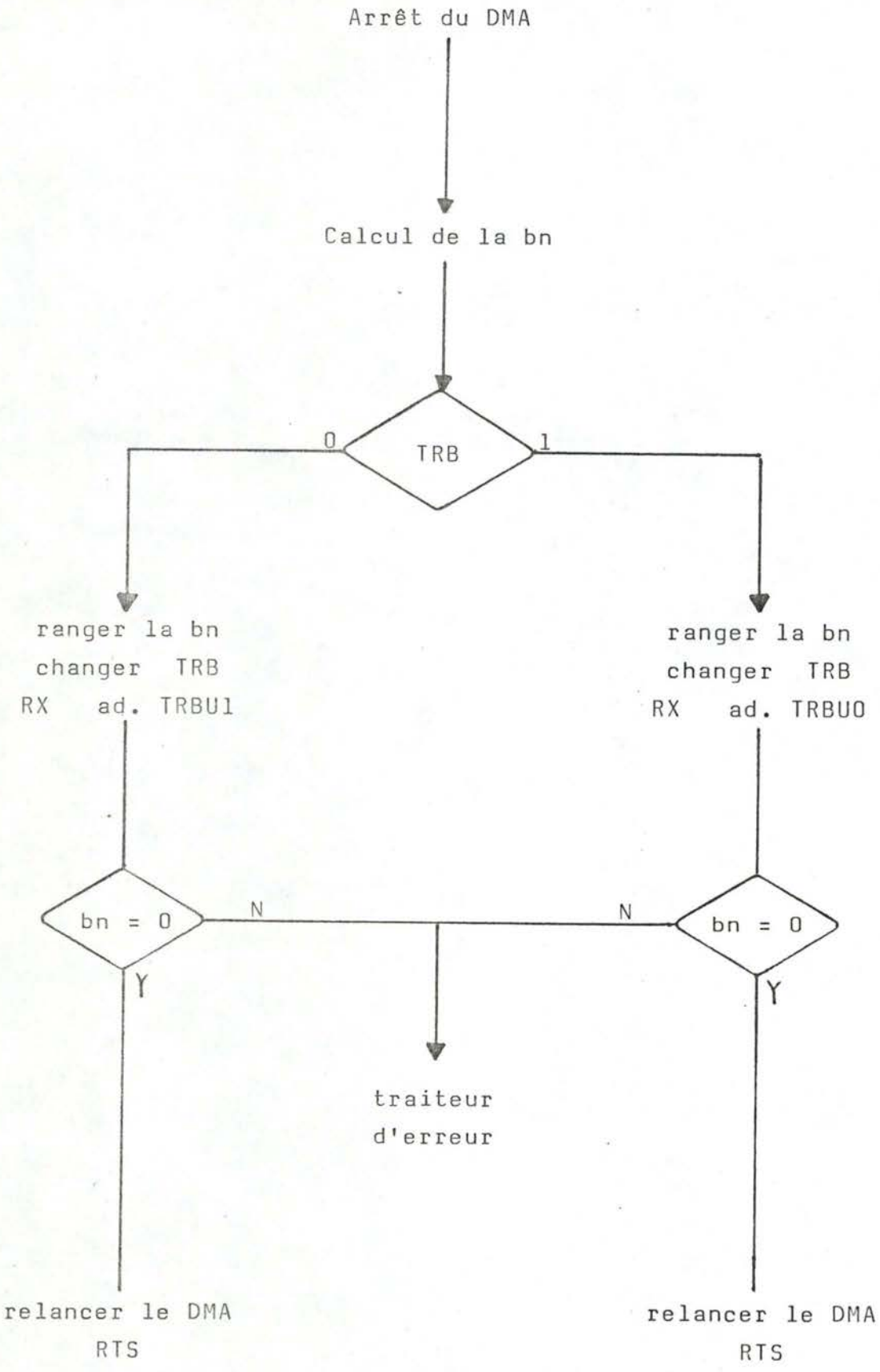


Fig. 3.8 - Module de fin de réception HDLC sans erreur

b) Fin de réception HDLC avec erreur

Dans ce cas, il suffit d'arrêter le DMA, de tester le buffer positionné et de relancer le DMA sur ce même buffer.

Ceci est schématisé à la figure 3.9.

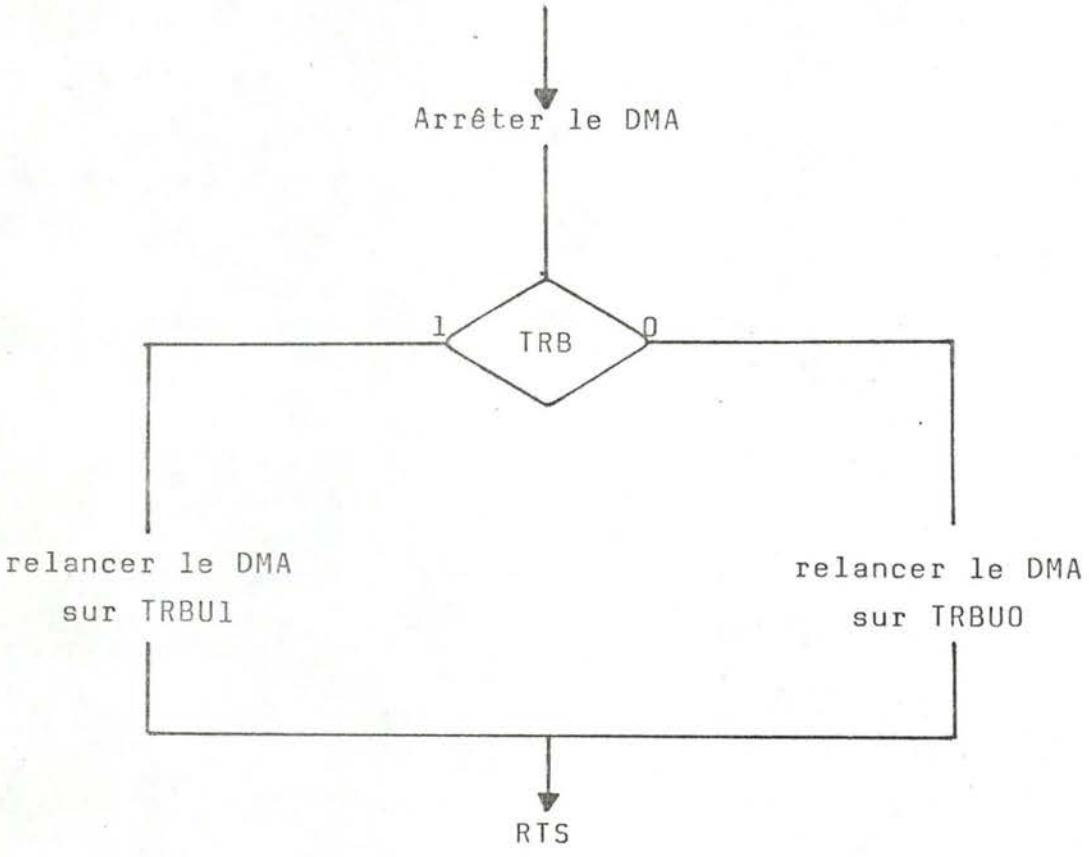


Fig. 3.9 - Module de fin de réception HDLC avec erreur

c) Fin de réception DMA

Ce cas aboutit à une routine d'erreur car il est théoriquement impossible. En effet, la longueur du DMA étant initialisée à une valeur supérieure à la longueur d'une trame, elle ne peut normalement pas s'annuler et ne donnera donc jamais un interrupt de fin de réception DMA. La justification du choix de cette longueur a été présentée au paragraphe 3.2.2.2. concernant l'initialisation du DMA.

3.5. Structure du sous-niveau trame 1 côté émission

3.5.1. Description de l'interface avec trame 2

L'automate de trame 2 contrôle le fonctionnement de l'émission à l'aide de deux zones.

La première, xmi - rq, lui permet de demander l'émission d'une trame de type S ou V. En effet, l'automate dépose dans cette variable le code correspondant à la trame à émettre :

xmi - rq =	01	RR
	02	RNR
	03	REJ
	05	SARM
	06	DISC
	07	UA
	08	CMDR

La seconde zone, xmi - cd, permet à trame 2 de contrôler le transfert des trames d'information effectué par trame 1. En effet, quand xmi - rq vaut 0 il y a essai d'émission de trame d'information. On peut distinguer quatre cas suivant xmi - cd :

xmi - cd = 00 : l'émission des trames d'information n'est pas autorisée.

= 01 : les trames d'information doivent être émises en séquence en respectant la gestion de la fenêtre.

= 02 : l'émission doit être relancée à partir de la première trame non encore acquittée (N(S) = V(N)). Dès que la commande est effectuée, xmi - cd est remis à 01.

= 03 : la première trame non encore acquittée doit être réémise avec le bit P mis à 1. En effet, ce cas correspond à une reprise sur temporisateur. Dès que la commande est effectuée, xmi - cd est remis à 00.

Du point de vue des buffers, l'interface est schématisé à la figure 3.10.

Le sous-niveau trame 2 range les paquets à émettre dans le buffer bu-out.

Trame 1 utilise les paquets qui sont dans bu-out pour garnir les trames d'informations. Il prépare ces trames dans le buffer xmi - bu afin de les expédier sur la ligne.

Il reste à trame 2 à enlever du buffer bu-out les paquets qui ont été acquittés.

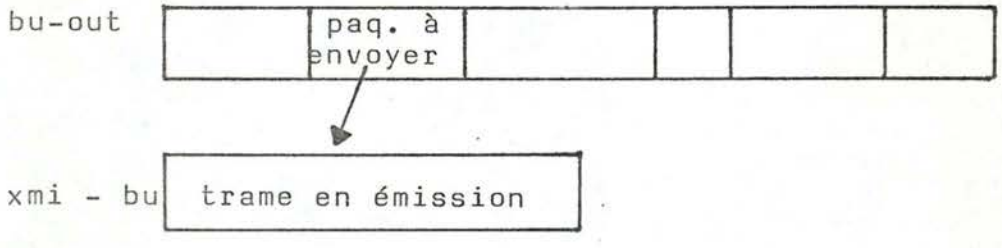


Fig. 3.10 - Buffers d'émission

3.5.2. Algorithmes

Comme pour la réception, nous avons ici aussi, trois cas à envisager : une fin d'émission HDLC sans erreur, une fin d'émission HDLC avec erreur, une fin d'émission DMA.

a) Fin d'émission HDLC sans erreur

La première opération à effectuer à la fin d'une émission est de tester la variable xmi - sv afin de savoir si c'était un flag envoyé ou non.

Si c'était un flag, nous essayons d'envoyer la trame suivante en allant à la routine CASFLG. Sinon, nous terminons la trame en cours en branchant en CASFCS après quoi nous allons aussi en CASFLG. La disposition de ces modules est représentée à la fig. 3.11.

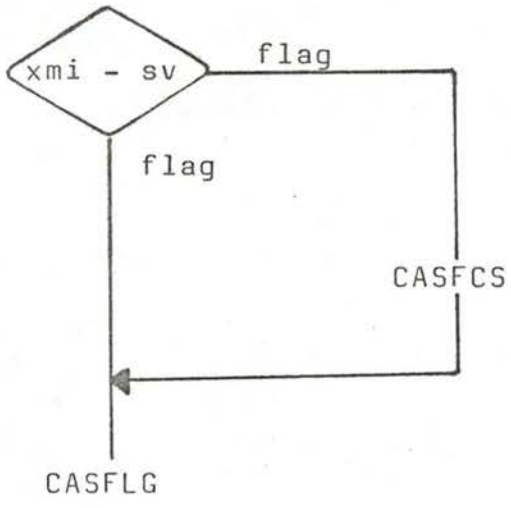


Fig. 3.11 - Disposition des modules en émission

1) Routine CASFCS

Cette routine se charge de terminer la trame qui était en cours. Il s'agit essentiellement de la mise à jour de variables. La figure 3.12 montre clairement les opérations réalisées.

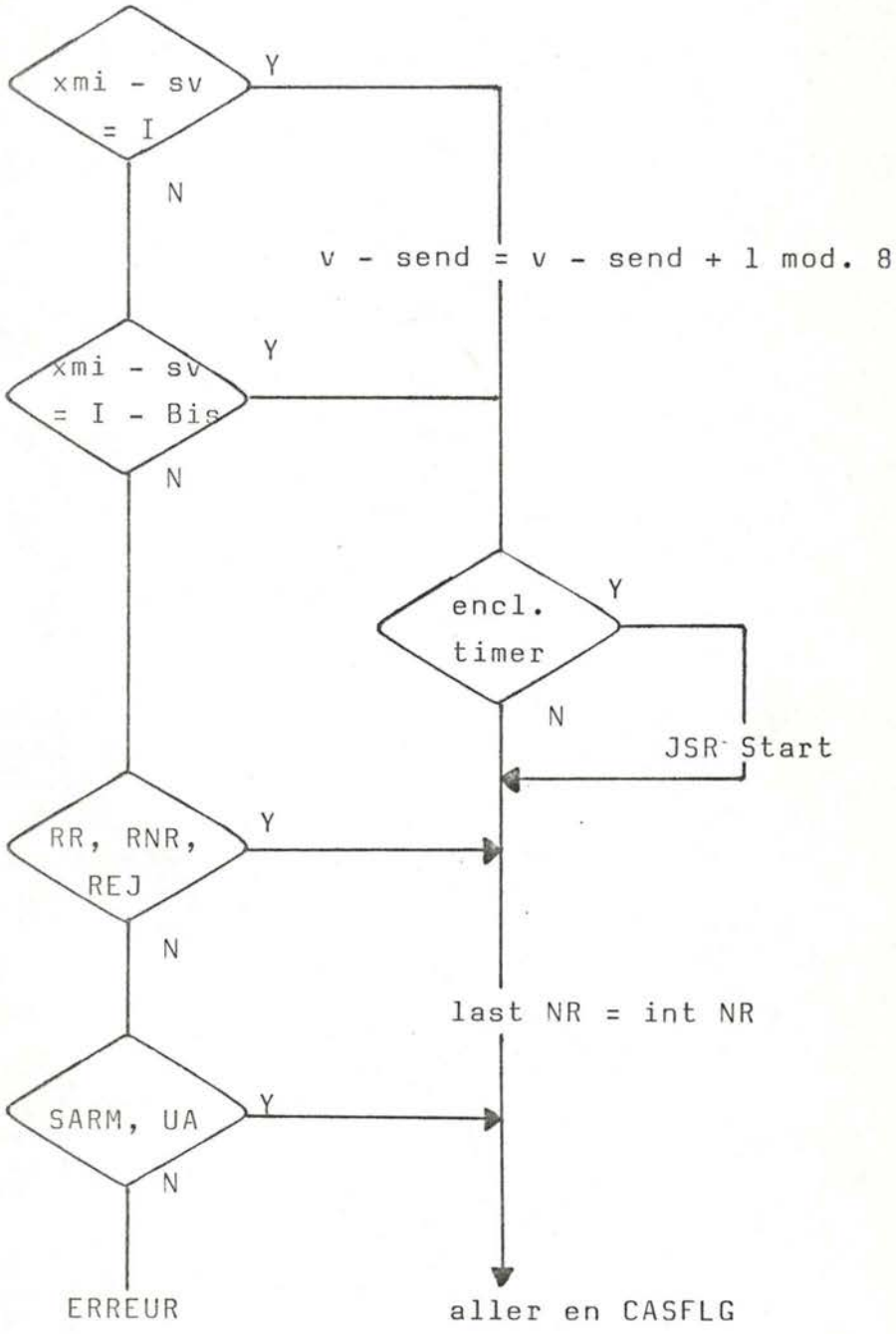


Fig. 3.12 - Module de terminaison de trame en cours

2) Routine CASFLG

Comme le montre la figure 3.13, ce groupe d'instructions effectue l'émission de la trame suivante.

En testant xmi - rq, nous pouvons savoir si on peut envoyer une trame autre qu'une trame d'information. Si oui, on construit dans xmi - bu la trame de type correspondant à la valeur de xmi - rq. Sinon, on essaie d'envoyer une trame d'information en allant au début de la routine de test d'émission d'une telle trame (ESSAI).

Il reste alors à préparer correctement le DMA pour l'émission proprement dite de la trame.

3) Routine ESSAI

Cette petite routine a comme fonction d'examiner si la trame d'information a pu être expédiée.

Si c'est le cas, on peut sortir. Sinon, on envoie un flag en positionnant le HDLC.

La figure 3.14 représente cet enchaînement.

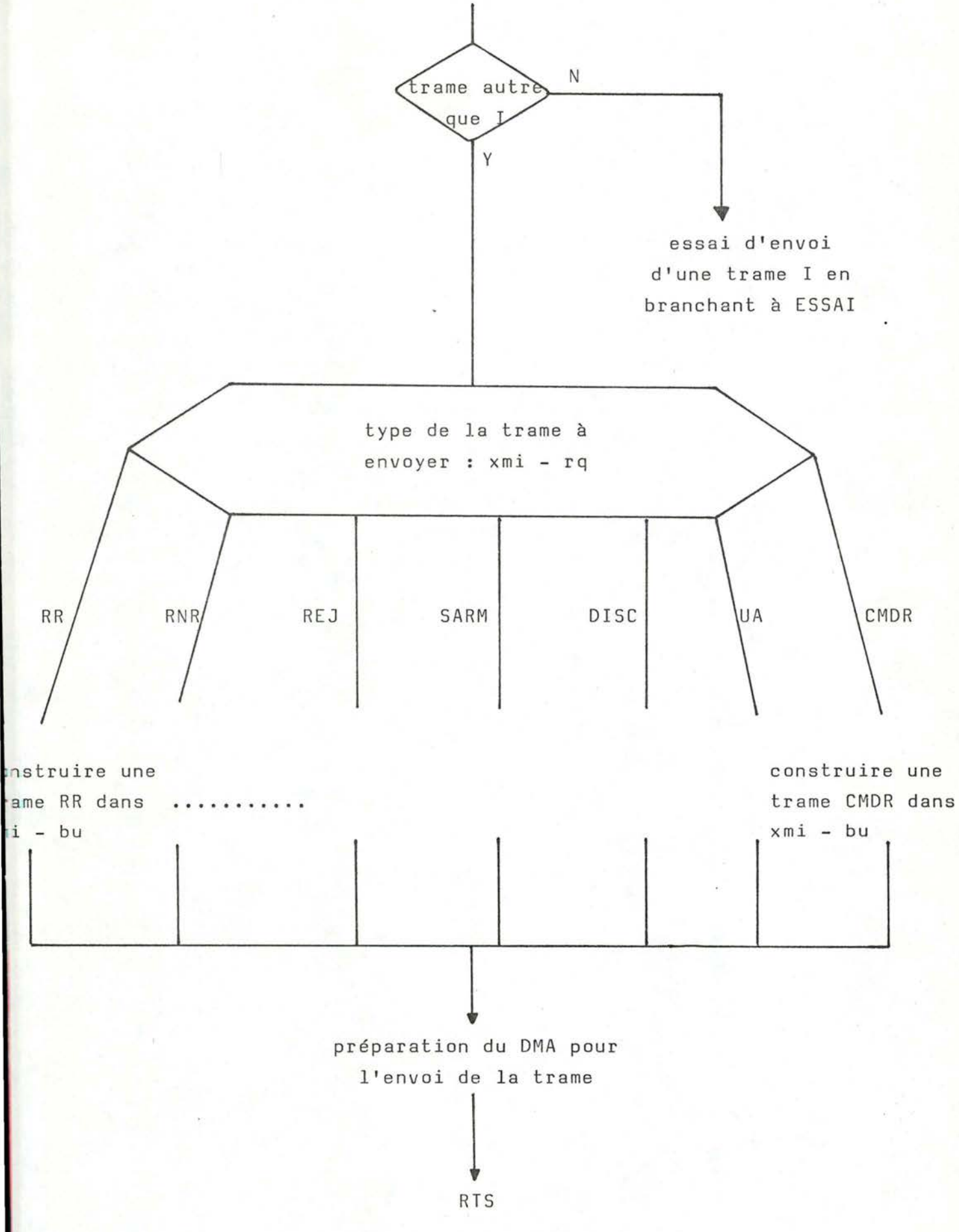


Fig. 3.1.3 - Module d'émission de la trame suivante

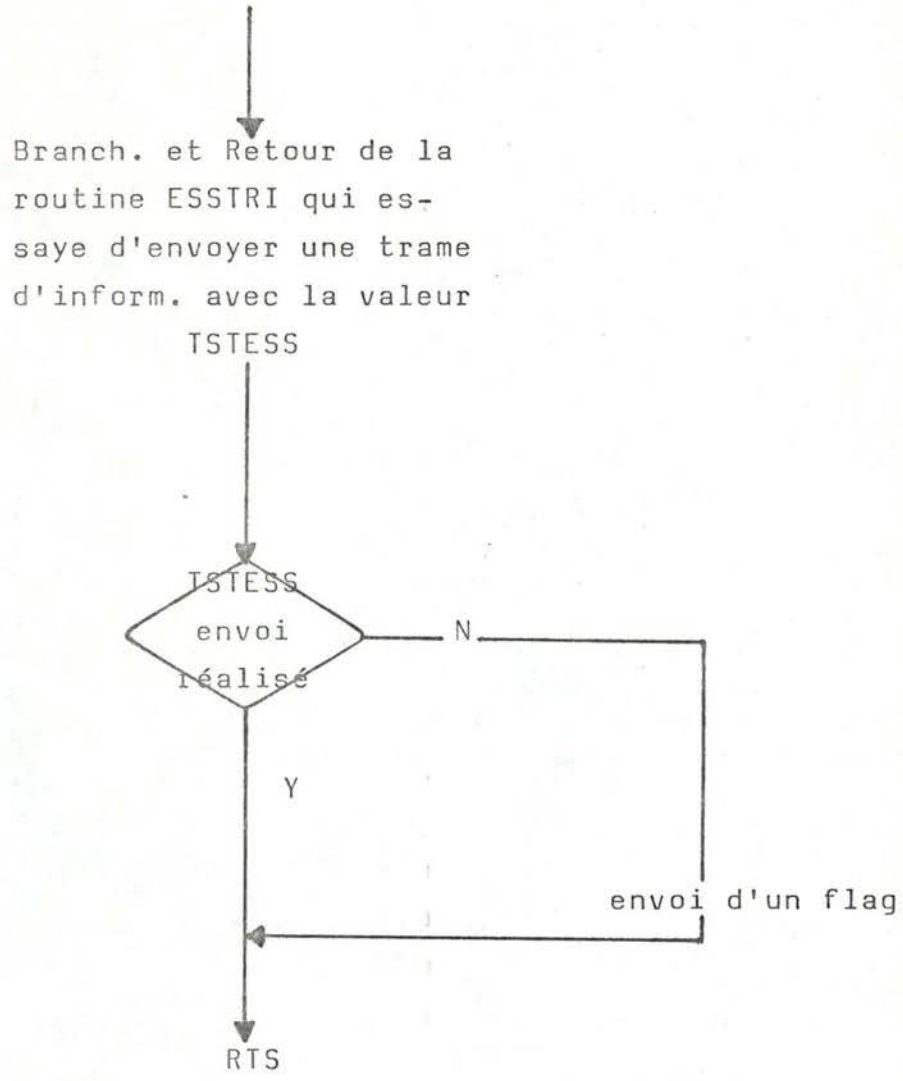


Fig. 3.14 - Module de test du résultat de l'émission d'une trame d'information

4) Routine ESSTRI (voir fig. 3.15)

La présente routine a pour but l'essai d'émission d'une trame d'information. Elle renvoie aussi la variable TSTESS qui signale à la routine ESSAI si l'envoi s'est bien passé ou non.

Comme indiqué au paragraphe 3.5.1. la variable xmi - cd peut prendre quatre valeurs.

a) Si xmi - cd vaut 00, l'émission n'est pas permise.

b) Si xmi - cd vaut 01, on est dans le cas de mode normal d'émission. S'il n'y a pas de paquet à émettre ou si on dépasse la fenêtre, il n'y a pas d'émission. Sinon, on envoie le paquet pointé par bo-opt. Pour cela, on met à jour les pointeurs et la fenêtre, on construit l'adresse et la commande avec les variables v - send et v - rec et on positionne correctement le DMA.

Il est à noter que si la longueur du nouveau paquet est nulle, le paquet suivant se trouve au début du buffer bu-out.

c) Si xmi - cd vaut 02, on est dans le cas de répétition c'est-à-dire que trame 2 a demandé la répétition à partir du paquet v-nack. Il suffit de positionner v-send et d'aller rechercher dans window le pointeur du paquet à renvoyer. Ceci effectué, on réalise l'émission comme dans le cas où xmi - cd vaut 01.

d) Si xmi - cd vaut 03, on a une reprise sur time-out. Une trame de numéro v-nack n'a pas été acquittée dans un temps déterminé. On reprend le paquet en allant rechercher son pointeur dans window et on l'envoie comme pour une émission normale mais avec le bit P mis à 1.

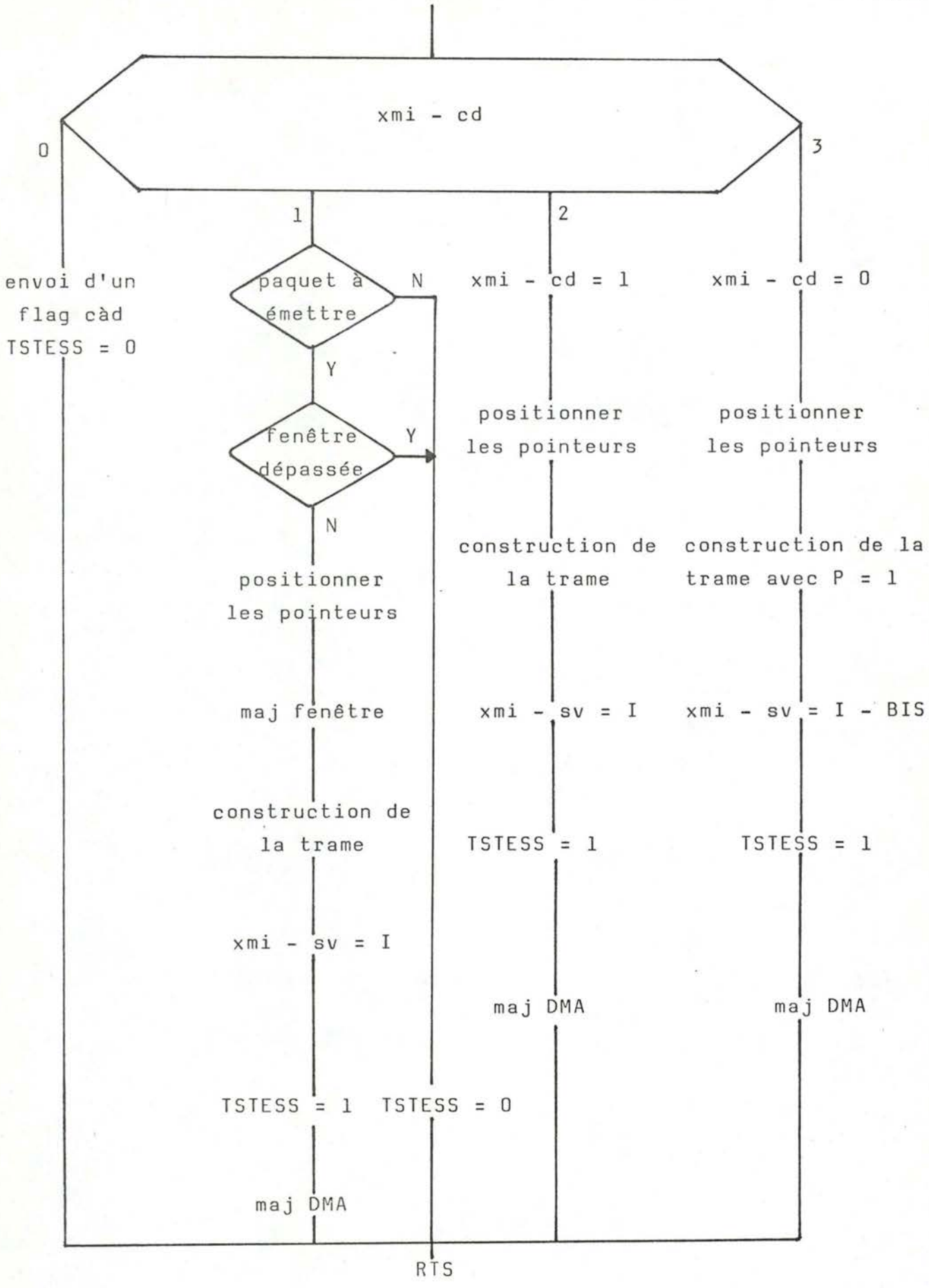


Fig. 3.15 - Module d'essai d'émission d'une trame d'information

b) Fin d'émission HDLC avec erreur

Ce cas constitue une erreur grave et amène à une routine de traitement d'erreur. En effet, aboutir à cette routine signifie que le HDLC n'a pas garni assez rapidement le THR. Ceci résulte donc d'un mauvais fonctionnement du hardware.

c) Fin d'émission DMA

Nous avons terminé l'émission d'une trame. Il reste à arrêter le DMA et à mettre l'HDLC en mode FCS afin de terminer l'émission.

Chapitre 4 - Procédures de tests

4.1. Introduction

Nous signalerons tout d'abord que les ouvrages "The Art of Software Testing" de Myers, "Program Test Methods" de Hetzel et "Program Style, Design, Efficiency, Debugging, and Testing" de Van Tassel repris dans la bibliographie concernant les procédures de tests ne traitent pas tellement des tests à effectuer lors de la mise en oeuvre d'un projet aussi important que la réalisation du niveau trame mais présentent plutôt différentes méthodes pour tester un programme.

En ce qui concerne le présent sujet, nous pouvons distinguer deux sortes de tests : les tests de conception et les tests de fonctionnement.

Les tests de conceptions sont des tests qui concernent une mise au point à long terme tant au niveau hardware que software. Ils permettent essentiellement de vérifier si les spécifications ont bien été comprises et si le système réalisé y répond correctement.

Afin de réaliser aisément tous ces tests, il est souhaitable de découper les programmes en différents modules (voir figure 4.1). Cette découpe permet de prévenir les erreurs au maximum et de détecter plus facilement celles qui restent en testant tout d'abord chaque module séparément puis les interfaces entre chacun d'eux.

Chacun des modules possédant une complexité fonctionnelle inférieure à celle du programme entier, il est avantageux de contrôler la fonction de chaque module séparément. En outre, ces tests seront effectués de manière telle que chaque instruction de chaque module soit exécutée au moins une fois.

Pour terminer le contrôle du bon fonctionnement du programme, il restera à tester tous les interfaces entre les modules qu'il contient.

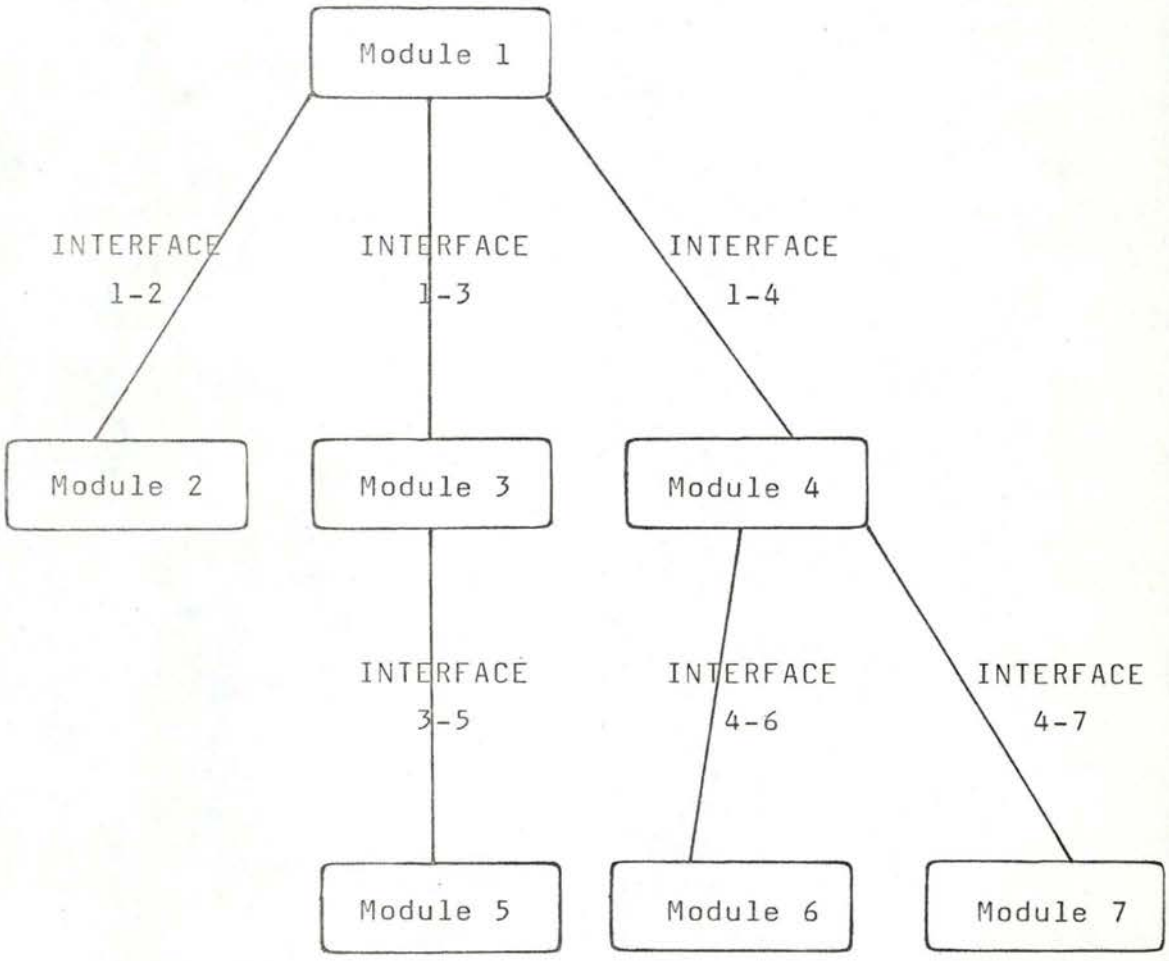


Fig. 4.1 - Découpage du programme en modules

Les tests de fonctionnements sont des tests de niveaux hardware et à court terme. Ce sont eux qui permettent de détecter la partie déficiente quand le système ne tourne plus. Ils ne peuvent être qu'hardware car le software, une fois mis au point, ne provoquera plus d'erreur. Par contre, le hardware peut amener des problèmes suite au mauvais fonctionnement d'un de ses composants.

Le but de ce chapitre est de présenter les procédures de tests réalisés pour le niveau trame. Il y sera réservé une place prépondérante pour les tests de conceptions des sous-niveaux trame 1 et trame 2. Nous traiterons le sous-niveau trame 2 en premier lieu afin de garder l'ordre chronologique des différents tests. En ce qui concerne les tests de fonctionnement, nous nous contenterons d'en présenter quelques-uns très sommairement.

4.2. Tests de conception

Le niveau trame étant découpé en deux sous-niveaux possédant des fonctions bien distinctes, il correspond parfaitement à l'idée de modularité.

Cette découpe a permis de tester les sous-niveaux trame 1 et trame 2 indépendamment avant de réaliser leur intégration.

Nous rappelons que les tests concernant le sous-niveau trame 2 et l'interface trame-paquet ont été réalisés l'année dernière par Monsieur Art.

Il restait cette année à tester le sous-niveau trame 1 et le niveau trame en entier c'est-à-dire l'intégration des deux sous-niveaux.

4.2.1. Tests de trame 2 seul

Ce chapitre a pour rôle d'expliciter le développement des tests concernant le sous-niveau trame 2 et d'expliquer le principe de SIMTRA.

a) Développement des tests

Il faut tout d'abord signaler que les microprocesseurs ont été construits par des étudiants de l'U.C.L. pendant la période des tests du sous-niveau trame 2. Ces deux mises au point ayant été effectuées en parallèle, il était impossible de tester les programmes de trame 2 directement sur les microprocesseurs.

Aussi, dans un premier temps, les tests ont-ils été réalisés sur un ordinateur Pdp 11/10. De plus, les programmes ont été écrits dans le langage C.

Ces programmes ont tout d'abord été testés séparément. Ils consistaient en :

- routines de gestion de la procédure du sous-niveau trame 2 : traitement des messages, des trames, du time-out;
- routines de gestion du DR 11-C pour l'échange de paquets et de messages sur l'interface trame-paquet;
- routines de gestion du DL 11-C pour la réception des caractères envoyés par le clavier du terminal de la pdp 11/10 et l'émission de caractères vers ce même terminal.

Ensuite, tous ces modules ont été regroupés pour former SIMTRA testé, lui aussi, sur la pdp 11/10. Nous rappelons à ce propos que la configuration de SIMTRA est représentée à la figure 2.6.

Dans un second temps, les programmes ont été traduits en assembleur par un cross-compilateur. Cette traduction a permis de tester SIMTRA en remplaçant la Pdp 11/10 par un microprocesseur Motorola 6800.

b) Principe de SIMTRA

SIMTRA se compose du côté de l'ETTD du sous-niveau trame 2 dans un microprocesseur, du niveau paquet dans la Pdp 11/45 et d'un terminal relié à la Pdp 11/45.

Le microprocesseur contient les programmes de trame 2 c'est-à-dire toute la gestion de l'automate, tandis que la Pdp 11/45 possède les programmes permettant d'initialiser ou de clôturer la liaison (OPEN, CLOSE) et d'envoyer ou de recevoir des paquets et des messages (PUT, GET).

Du côté de l'ETCD, SIMTRA contient un terminal connecté au microprocesseur, permettant de simuler à la main le sous-niveau trame 2.

Un opérateur installé au terminal relié à la Pdp 11/45 peut envoyer (recevoir) des paquets vers le (du) terminal connecté au microprocesseur. Les paquets ainsi reçus sont affichés sur l'écran.

Un autre opérateur travaillant au terminal relié au microprocesseur peut envoyer (recevoir) des trames vers le (du) terminal connecté à la Pdp 11/45. Les trames envoyées sont mémorisées en minuscules sur l'écran tandis que celles reçues y sont affichées en majuscules. De plus, à tout moment, l'état de l'automate est indiqué sur cet écran.

Cette configuration a permis d'effectuer les tests complets des fonctions de l'automate. En effet, pour chacun de ses états, il suffit de générer tous les événements possibles et de vérifier pour chacun d'eux si la transition et les actions de l'automate sont correctes.

Pour réaliser concrètement ce test, il faut aller chercher les programmes dans les directories adéquates. Le schéma de la structure de ces directories est présenté à la figure 5.3.

La version exécutable du programme tournant dans le microprocesseur se trouve dans la directory/ ARTLAMB/ TRAME 2/ TRA 268/ EXE 268 sous le nom SIMTR2.

Les programmes à faire tourner sur la pdp 11/45 sont dans la directory/ ARTLAMB/ ART/ SIM 68/ et ont pour noms OPEN 1, OPEN 2, CLOSE 1, CLOSE 2, GET 1, GET 2, PUT 1, PUT 2. Ces programmes servent à simuler le niveau paquet. Voici brièvement leurs fonctions :

- OPEN envoie un message de demande de connexion;
- CLOSE envoie un message de demande de déconnexion;
- GET permet de visualiser sur le terminal connecté à la pdp 11/45 tout ce qui vient du niveau trame vers le niveau paquet;
- PUT permet d'envoyer des paquets du niveau paquet vers le niveau trame.

4.2.2. Tests de trame 1 seul

Ce chapitre présente l'évolution des tests effectués à propos du sous-niveau trame 1.

Il contient tout d'abord une série de tests préliminaires vérifiant certains procédés utilisés par trame 1 tels que le X21 bis, l'HDLC et le DMA.

La deuxième partie explicite les tests des routines effectives du sous-niveau trame 1.

4.2.2.1. Tests préliminaires

4.2.2.1.1. Tests de la connection des 2 microprocesseurs

Le rôle du sous-niveau trame 1 est de permettre le passage de trames d'un microprocesseur vers un autre. Le premier test à effectuer est donc de tester la bonne connection entre les deux microprocesseurs c'est-à-dire le niveau physique ou X21 bis.

Nous ne reprendrons pas dans ce paragraphe le fonctionnement du X21 bis; celui-ci a été présenté en détail au chapitre 3.2.1. Il faut cependant remarquer que le microprocesseur ETCD possède un PIA (PIA ETCD) que n'a pas l'ETTD. En effet, le microprocesseur ETTD simule un terminal tandis que le microprocesseur ETCD simule un modem. Il en résulte une dissymétrie du point de vue X21 bis car le modem possède des lignes que n'a pas le terminal. Le "chip" HDLC est identique pour les deux microprocesseurs et comprend les fonctions de l'ETTD. Le PIA ETCD simule les lignes du modem pour l'ETCD.

Lors de l'initialisation du X21 bis, il faudra donc programmer le PIA ETCD de façon telle que les lignes qu'il simule soient correctement positionnées.

Le schéma 4.2 représente la configuration utilisée pour réaliser le test : dans le microprocesseur ETCD tourne un programme initialisant le PIA ETCD et le X21 bis du côté de l'ETCD tandis que dans le microprocesseur ETTD fonctionne un programme initialisant le X21 bis du côté de l'ETTD.

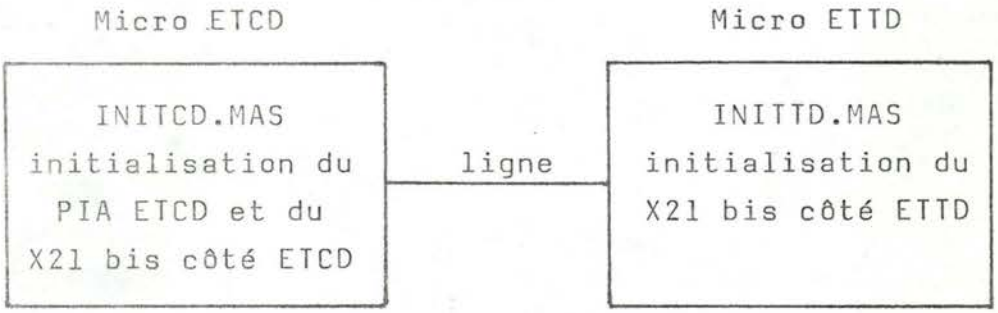


Fig. 4.2 - Test de l'établissement de la fonction physique

Pour effectuer ce test, on peut aller chercher les versions exécutables des programmes INITCD.MAS et INITTD.MAS dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms INITCD et INITTD.

L'établissement de la jonction physique peut être vérifié grâce à l'allumage des lampes se trouvant devant les microprocesseurs. Ce contrôle est clairement expliqué aux pages 56 et 57 du mémoire de Messieurs Beudin et Zone.

4.2.2.1.2. Test du fonctionnement du HDLC

La connection entre les deux microprocesseurs étant réalisée, il faut maintenant s'assurer du bon fonctionnement des outils utilisés pour le transfert.

Le premier de ces outils est l'HDLC. Son mode d'emploi du point de vue de la programmation se trouve en annexe A.

a) Test du HDLC sans mode auto-flag

L'HDLC a tout d'abord été testé dans sa forme la plus simple : sans mode auto-flag, c'est-à-dire que le programmeur doit lui-même envoyer les flags par une commande software.

Ce test est représenté à la figure 4.3 : un programme tournant dans le microprocesseur ETTD envoie sur la ligne au moyen du HDLC programmé sans mode auto-flag une trame qui sera reçue par un programme tournant dans le microprocesseur ETCD.

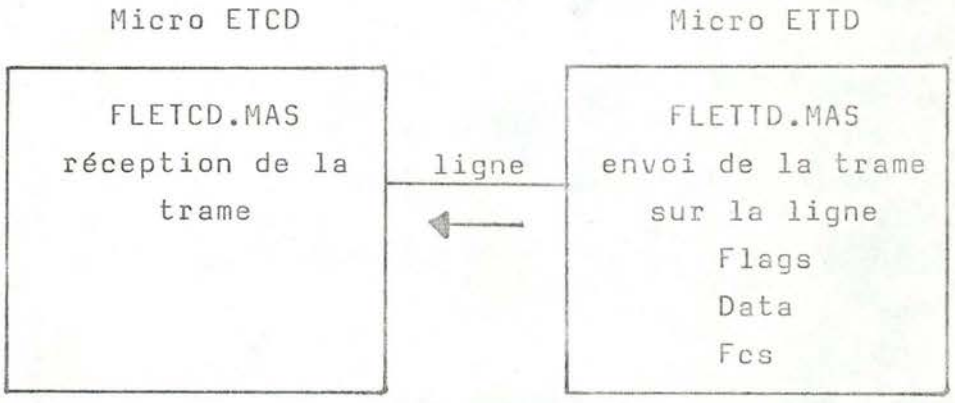


Fig. 4.3 - Test du fonctionnement du HDLC sans mode auto-flag dans le sens ETTD - ETCD

Les tests de ce mode du HDLC n'ont été réalisés que dans le sens ETTD - ETCD car le HDLC, dans la version finale de trame 1, sera utilisé avec le mode auto-flag.

Les versions exécutables des programmes FLETCD.MAS et FLETTD.MAS se trouvent dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms FLETCD et FLETTD.

Le bon fonctionnement du HDLC peut être vérifié en allant contrôler que le contenu du buffer de réception du programme FLETCD.MAS est le même que le contenu du buffer d'émission du programme FLETTD.MAS.

b) Test du HDLC avec mode auto-flag

L'HDLC a ensuite été testé avec l'utilisation du mode auto-flag. Quand cette technique est employée, le programmeur n'a plus à s'inquiéter de l'envoi des flags entre les trames car celui-ci est réalisé automatiquement par l'HDLC.

Pour ce mode auto-flag, les tests ont été réalisés dans les deux sens car, comme nous l'avons signalé ci-dessus, c'est lui qui sera utilisé dans la version terminale de trame 1.

Les figures 4.4 et 4.5 nous présentent les programmes utilisés ainsi que le sens des transferts.

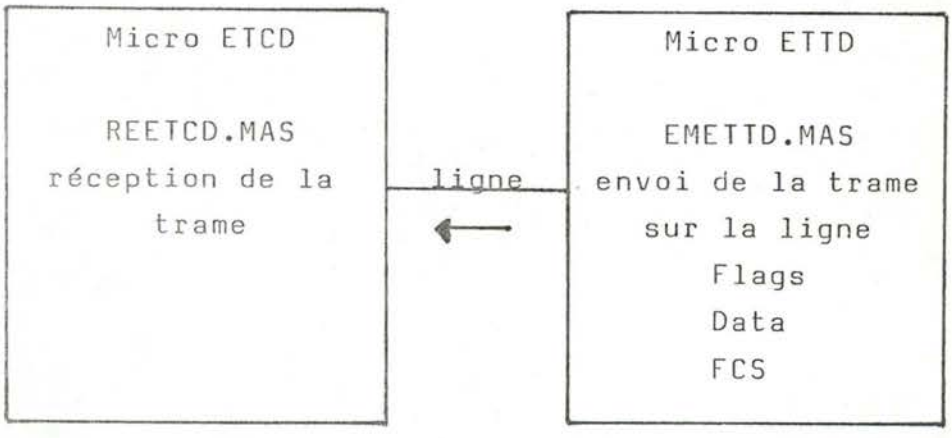


Fig. 4.4 - Test du fonctionnement du HDLC avec mode auto-flag dans le sens ETTD - ETCD

Pour effectuer le test dans le sens ETTD - ETCD (figure 4.4) on peut aller chercher les versions exécutables des programmes REETCD.MAS et EMETTD.MAS dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms REETCD et EMETTD.

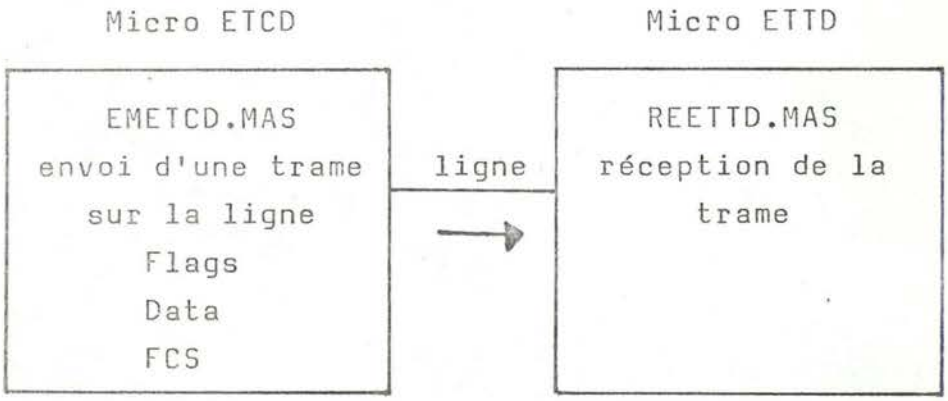


Fig. 4.5 - Test du fonctionnement du HDLC avec mode auto-flag dans le sens ETCD - ETTD

Pour réaliser le test du transfert du microprocesseur ETCD vers le microprocesseur ETTD (figure 4.5), on peut aller chercher les versions exécutables des programmes EMETCD.MAS et REETTD. MAS dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms EMETCD et REETTD.

Le principe de la vérification du bon fonctionnement de ces tests est le même que pour le HDLC sans mode autoflag : vérifier que le contenu du buffer d'émission est le même que celui du buffer de réception.

4.2.2.1.3. Test du fonctionnement du DMA

Le second outil utilisé pour les transferts de trame est le DMA. Les indications quant à sa programmation se trouvent à l'annexe B.

Nous allons expliquer en détail la vérification du fonctionnement correct du DMA dans le sens ETCD - ETTD; pour l'autre sens, les tests étant similaires, nous nous contenterons de présenter les schémas.

La vérification du bon fonctionnement du DMA s'est effectuée en trois temps représentés aux figures 4.6, 4.7 et 4.8.

Le premier temps consiste à tester le DMA en réception (figure 4.6). Le programme EMETCD.MAS émettant au moyen du HDLC étant correct, nous pouvons l'utiliser pour émettre de l'ETCD et tester ainsi le programme DMATD3.MAS dont la fonction est de recevoir sur l'ETTD et en DMA les trames venant de la ligne.

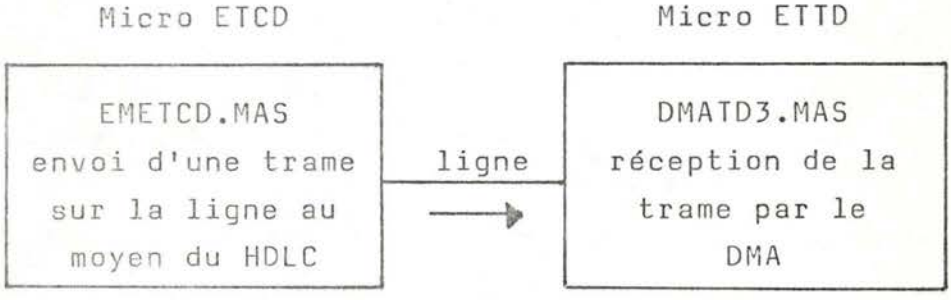


Fig. 4.6 - Test du fonctionnement du DMA en réception dans le sens ETCD - ETTD

Le second temps consiste à tester le DMA en émission en utilisant le programme correct REETTD.MAS en réception. Ceci permet de mettre au point le programme DMACD3.MAS qui émet en DMA de l'ETCD. La figure 4.7 schématise cette deuxième partie du test.

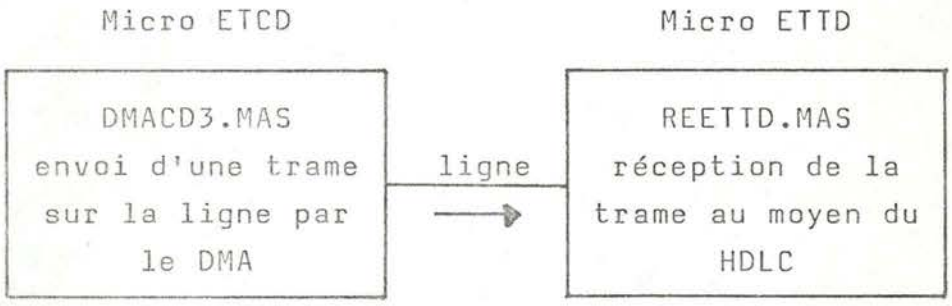


Fig. 4.7 - Test du fonctionnement du DMA en émission dans le sens ETCD - ETTD

Les deux programmes DMATD3.MAS et DMACD3.MAS ayant été testés, la confirmation de leur bon fonctionnement peut être effectuée par la configuration décrite à la figure 4.8.

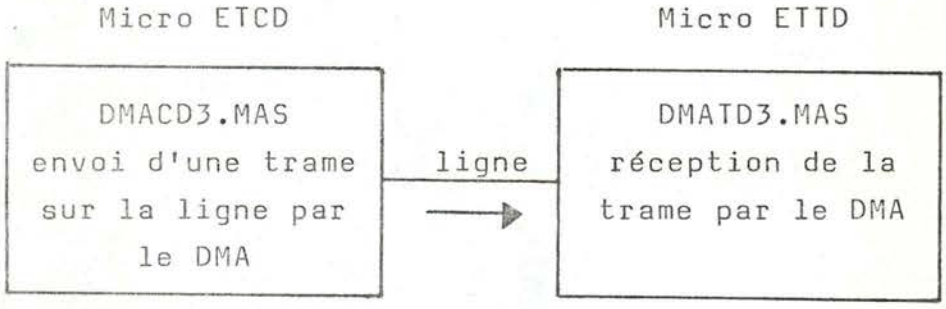


Fig. 4.8 - Test du fonctionnement du DMA en émission et réception dans le sens ETCD - ETTD

Les versions exécutables des programmes DMACD3.MAS et DMATD3.MAS se trouvent dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms DMACD3 et DMATD3.

La séquence des tests réalisés pour vérifier le DMA dans le sens ETTD - ETCD est présentée aux figures 4.9, 4.10 et 4.11.

Ces tests permettent de vérifier les programmes DMACD1.MAS et DMATD1.MAS dont les versions exécutables sont dans la directory/ ARTLAMB/ TRAME 1/ PRELIM/ EXEPRE sous les noms DMACD3 et DMATD3.

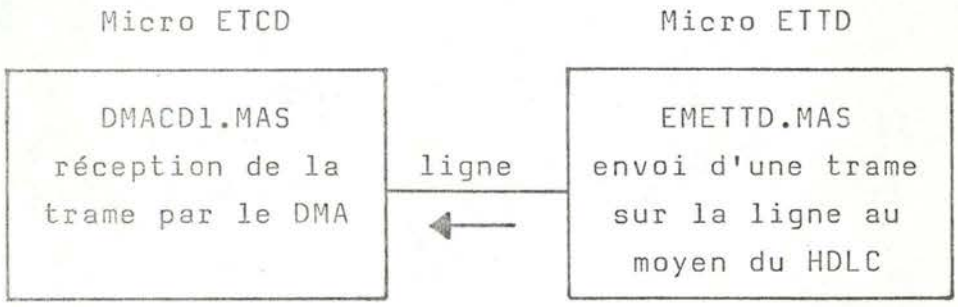


Fig. 4.9 - Test du fonctionnement du DMA en réception dans le sens ETTD - ETCD

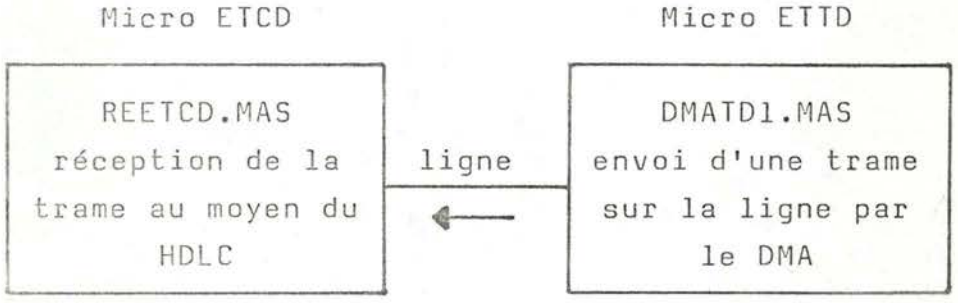


Fig. 4.10 - Test du fonctionnement du DMA en émission dans le sens ETTD - ETCD

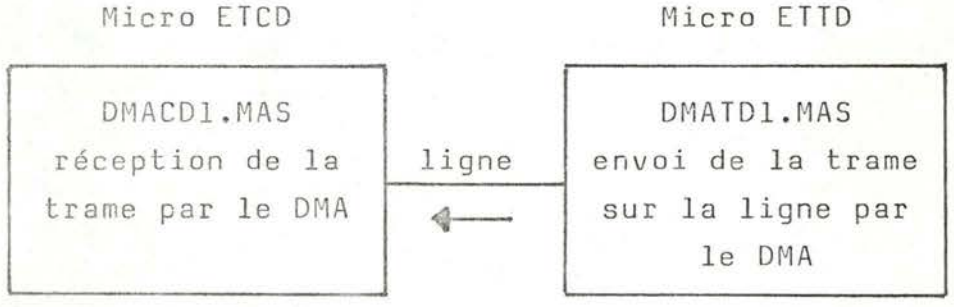


Fig. 4.11 - Test du fonctionnement du DMA en émission et réception dans le sens ETTD - ETCD

4.2.2.2. Tests de trame 1

Tous les outils utilisés par trame 1 et servant aux transferts des trames ayant étés mis au point, les tests effectifs du sous-niveau trame 1 peuvent commencer. Ils vont s'effectuer dans les deux sens et pour chacun d'eux nous procéderons à quatre contrôles : la réception d'une trame, la réception de deux trames, l'émission d'une trame et l'émission de deux trames.

4.2.2.2.1. Tests de la réception

a) Réception d'une trame

Le premier test effectué est celui de la réception d'une trame. Pour son émission, nous utiliserons des programmes déjà testés : DMATD1. MAS qui émet du microprocesseur ETTD et DMACD3. MAS qui émet du microprocesseur ETCD.

Ce test va nous permettre de contrôler une partie de la boucle d'interruption ainsi qu'un morceau de la routine de réception sans erreur.

Les programmes utilisés ainsi que les sens des transferts sont représentés aux figures 4.12 et 4.13. Les versions exécutables des programmes TRRECD.MAS et TRRETD.MAS qui reçoivent respectivement sur l'ETCD et sur l'ETTD se trouvent dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les noms TRRECD et TRRETD.

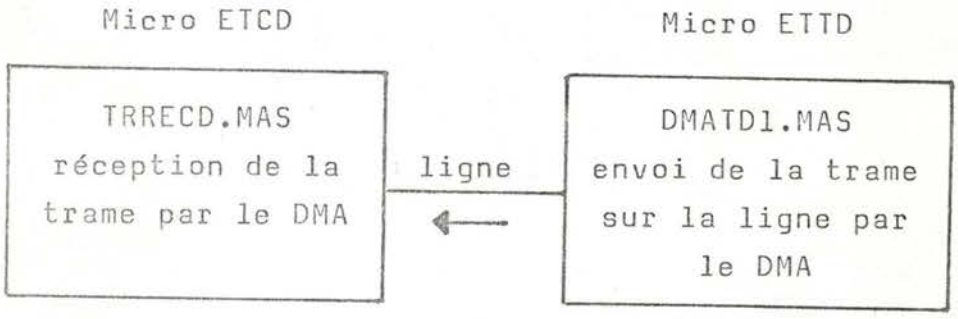


Fig. 4.12 - Test de la réception d'une trame du sous-niveau trame 1 pour l'ETCD

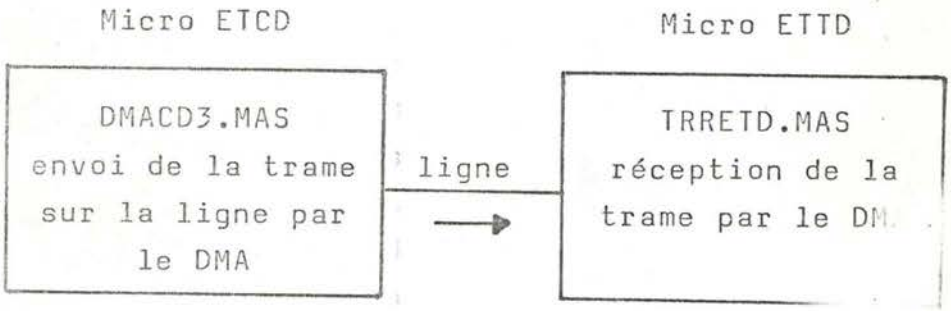


Fig. 4.13 - Test de la réception d'une trame du sous-niveau trame 1 pour l'ETTD.

b) Réception de deux trames

Afin de tester toute la dynamique de la routine de réception sans erreur, il faut envoyer deux trames vers le sous-niveau trame 1. En effet, la réception se fait dans un double buffer et utilise des indicateurs prenant alternativement deux valeurs lors des passages dans la routine. Toutes les possibilités n'auront donc été vérifiées qu'après deux exécutions du module de réception.

Pour réaliser ces contrôles, les programmes de réception utilisés pour le test précédent viennent encore ici. Pour l'émission des trames,

nous utiliserons les programmes DMATD4.MAS et DMACD4.MAS qui envoient successivement deux trames sur la ligne, respectivement de l'ETTD et de l'ETCD. On peut aller chercher leur version exécutable dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les noms DMATD4 et DMACD4.

Les figures 4.14 et 4.15 ci-après schématisent les deux tests réalisés et rappellent les programmes utilisés.

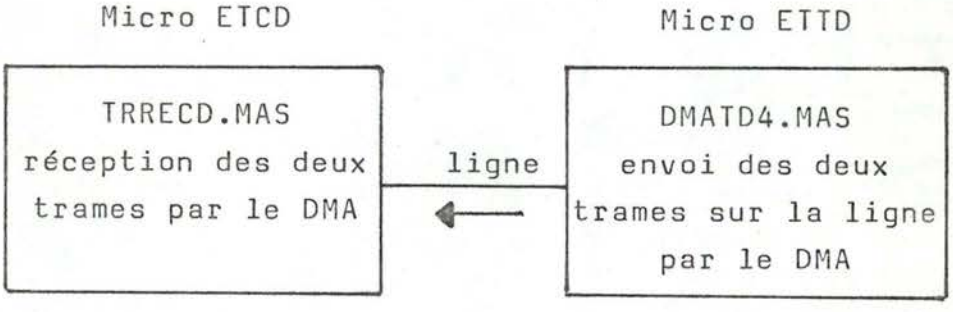


Fig. 4.14 - Test de la réception de deux trames du sous-niveau trame 1 pour l'ETCD

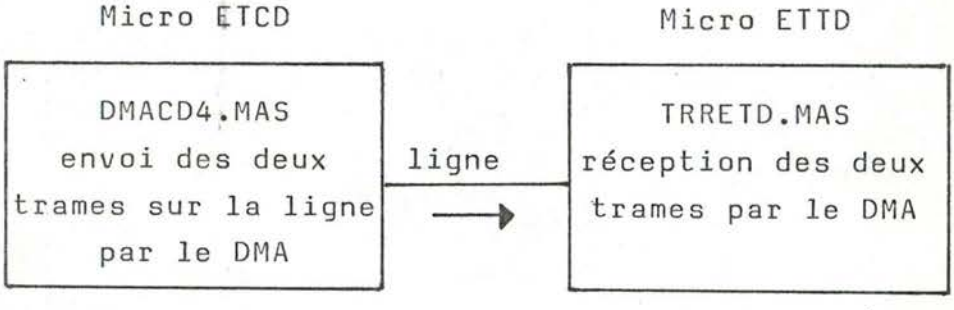


Fig. 4.15 - Test de la réception de deux trames du sous-niveau trame 1 pour l'ETTD

4.2.2.2.2. Tests de l'émission

Les tests de la réception étaient assez réduits car la routine exécutée est la même quelque soit la trame à recevoir.

L'émission d'une trame, par contre, implique le passage par une routine différente pour chacun de ses types car c'est le sous-niveau trame 1 qui construit la trame à envoyer. Cette construction entraîne l'écriture d'un programme de test différent par type de trame à émettre.

a) Emission d'une trame

On peut considérer qu'il y a onze types différents d'envoi de trame. Nous aurons donc onze programmes d'émission à tester.

Les types de trames à envoyer sont les suivants : SARM, DISC, UA, CMDR, RR, RNR, REJ ainsi que quatre sortes de trame d'information suivant la valeur de la variable xmi - cd.

Les programmes qui recevront les trames seront les programmes DMATD3.MAS pour le microprocesseur ETTD et DMACD1.MAS pour le microprocesseur ETCD, déjà testés lors de la vérification du DMA. Les versions exécutables se trouvent dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les noms DMATD3 ET DMACD1.

L'émission de ces différentes trames va nous permettre de contrôler les séquences non encore testées de la boucle d'interruption ainsi que toutes les routines réservées à la construction et à l'envoi des trames.

Les figures 4.16 et 4.17 nous montrent les programmes utilisés pour tester un type de trames (SARM) dans les deux sens. Les différents programmes utilisés pour l'émission de l'ETCD sont les suivants : SREMCD.MAS, DIEMCD.MAS, UAEMCD.MAS,

CREMCD.MAS, RREMCD.MAS, RNEMCD.MAS, RJEMCD.MAS, I1EMCD.MAS, I2EMCD.MAS, I3EMCD.MAS et I4EMCD.MAS pour émettre respectivement une trame SARM, DISC, UA, CMDR, RR, RNR, REJ et d'information avec xmi - cd prenant les valeurs un à quatre. Pour émettre de l'ETTD, les noms sont identiques; il suffit de remplacer "CD" par "TD". Les versions exécutables de ces programmes sont dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les mêmes noms que les versions sources excepté le ".MAS".

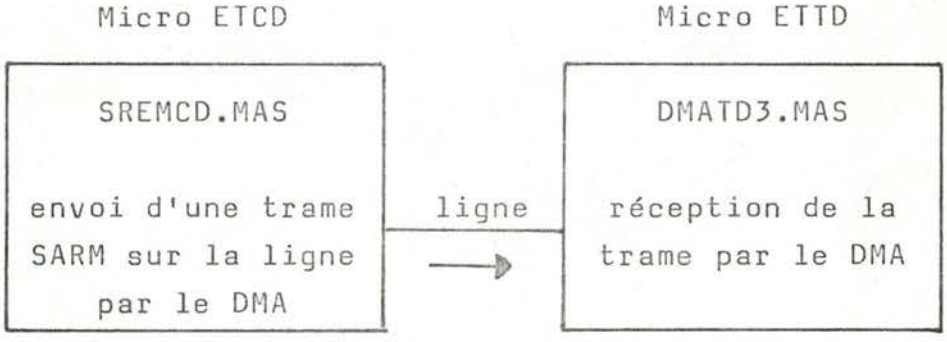


Fig. 4.16 - Test de l'émission d'une trame SARM du sous-niveau trame 1 pour l'ETCD

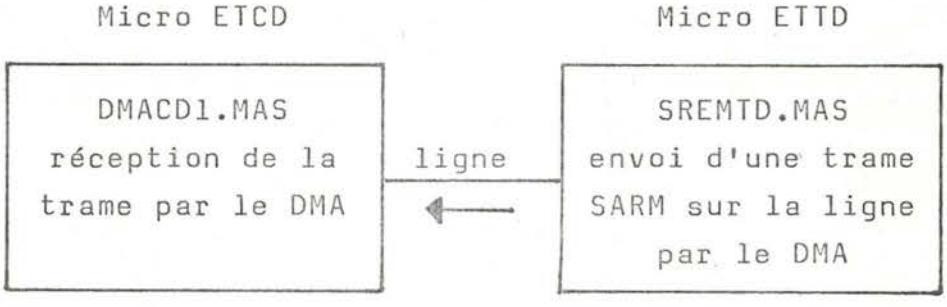


Fig. 4.17 - Test de l'émission d'une trame SARM du sous-niveau trame 1 pour l'ETCD

La manière de réaliser ces tests est la suivante : on met tout d'abord l'HDLC en mode "émission de flags" en positionnant le CR1 et on met le code FLAG dans xmi - sv. Cette opération va nous permettre d'émettre un FLAG puis d'entrer dans la routine de fin d'émission sans erreur (XMNOER).

Pour chaque type de trame, on aura aussi positionné les différentes variables servant à son émission afin d'exécuter les routines adéquates pour la construction et de mettre dans le buffer d'émission les valeurs correctes à émettre.

Pour un SARM, il suffit de mettre le code SARM dans la variable qui donne le type de trame à émettre (xmi - rq) et de mettre xmi - cd à zéro. Les opérations sont identiques pour toutes les trames de commande.

Pour une trame d'information, il faut préparer correctement toutes les variables servant à son émission : mettre xmi - rq à la valeur 0, xmi - cd à la valeur du type de trame d'information à envoyer, mettre dans v - send, v - nack et v - rec le numéro des dernières trames respectivement envoyées, acquittées et reçues, positionner le compteur de paquet, bo - pct, à 1 et construire la trame dans le buffer bu-out.

Après l'exécution de la routine de fin d'émission sans erreur, ce test va nous permettre de contrôler la routine de fin d'émission DMA chargée de [illegible]r la trame par un FCS.

b) Emission de 2 trames

Les programmes de tests utilisés dans ce paragraphe assurent l'émission de deux trames consécutives.

Un premier programme envoie une trame SARM suivie d'une trame UA. Cette configuration, présentée aux figures 4.18 et 4.19 permet de simuler un établissement de la connection.

Les programmes utilisés pour l'émission ont pour noms TREMCD.MAS et TREMTD.MAS. Leur version exécutable se trouvent dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les noms TREMCD et TREMTD. Les programmes utilisés en réception sont ceux testés au paragraphe 4.2.2.2.1.

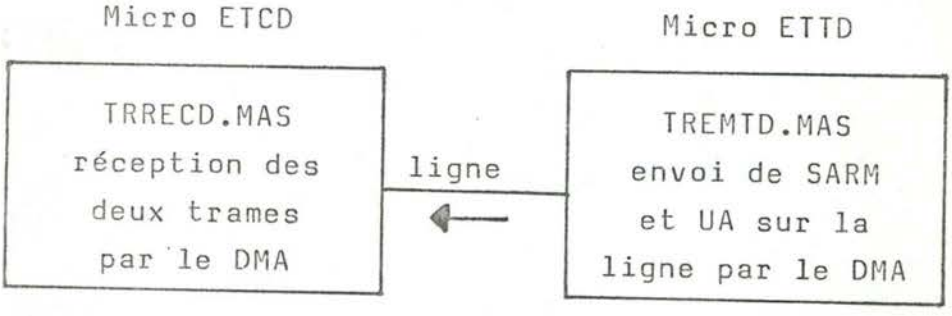


Fig. 4.18 - Test de l'émission de deux trames du sous-niveau trame 1 pour l'ETTD

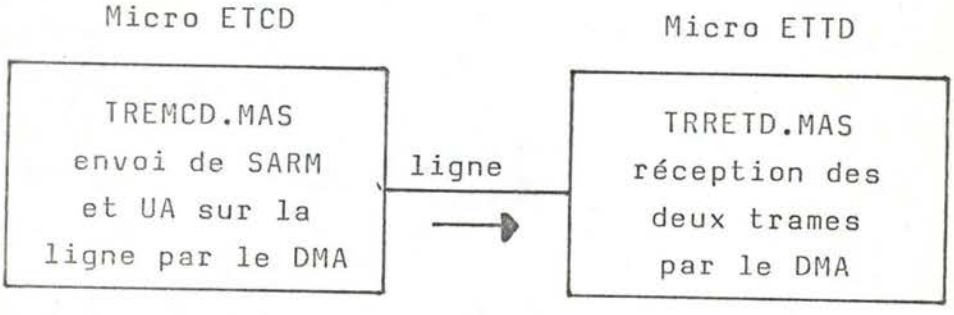


Fig. 4.19 - Test de l'émission de deux trames du sous-niveau trame 1 pour l'ETCD

Le test suivant effectue l'émission de deux trames d'information consécutives. Il utilise les programmes INCD22.MAS et INTD22.MAS dont les versions exécutables se trouvent dans la directory/ ARTLAMB/ TRAME 1/ TRA 1/ EXETR 1 sous les noms INCD22 et INTD22. Les figures 4.20 et 4.21 nous exposent la configuration de ce test.

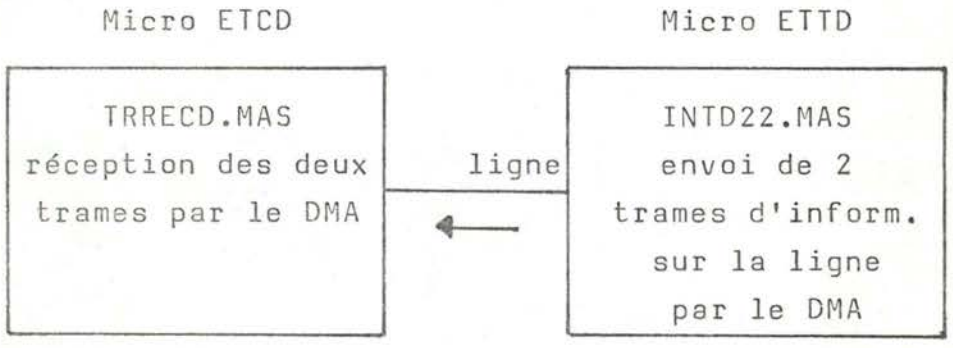


Fig. 4.20 - Test de l'émission de deux trames d'information du sous-niveau trame 1 pour l'ETTD

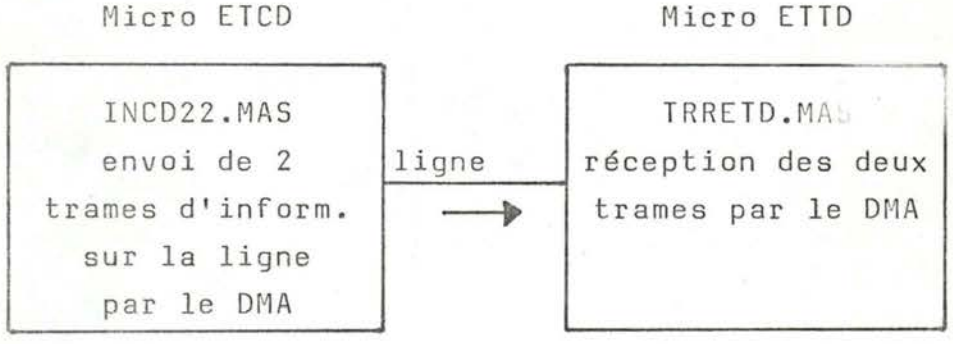


Fig. 4.21 - Test de l'émission de deux trames d'information du sous-niveau trame 1 pour l'ETCD

Ces tests permettent aussi de tester la dynamique de la boucle d'interruption puisqu'ils imposent l'exécution des mêmes routines à plusieurs reprises. De plus, l'évolution des pointeurs, variables et buffers peut également être contrôlée par ces programmes.

4.2.3. Test de l'intégration

Les tests de l'intégration des deux sous-niveaux trame 1 et trame 2 ont été effectués en deux temps.

La première partie a été consacrée à la vérification de l'initialisation de la liaison tandis que la seconde a contrôlé l'intégration proprement dite.

4.2.3.1. Test de l'initialisation de la liaison

Ces tests ont été réalisés dans les deux sens. Nous n'en présenterons qu'un, l'autre étant tout à fait analogue.

La configuration présentée à la figure 4.22 montre l'initialisation lancée à partir du microprocesseur ETTD.

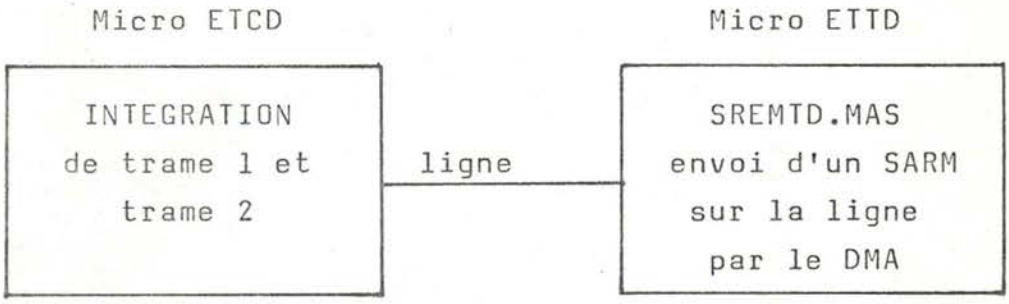


Fig. 4.22 - Test de l'initialisation de l'intégration à partir de l'ETTD

Le programme SREMTD.MAS tournant dans le microprocesseur ETTD envoie un SARM sur la ligne. L'intégration de trame 1 et trame 2 qui tourne dans le microprocesseur ETCD le reçoit et réagit en émettant une trame UA suivie d'un SARM.

Ceci permet de tester trame 1 à la fois en émission et en réception et de vérifier en partie la validité des interfaces entre les sous-niveaux trame 1 et trame 2.

4.2.3.2. Test de l'intégration proprement dite

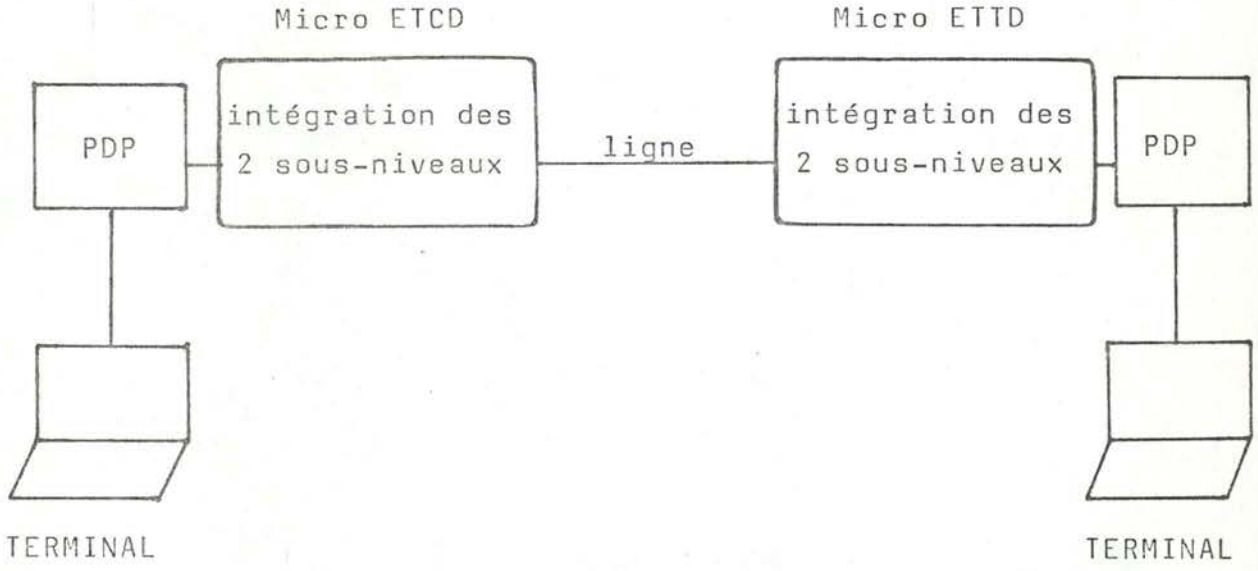


Fig. 4.23 - Test de l'intégration des 2 sous-niveaux

L'initialisation étant correcte, il reste à s'assurer du bon fonctionnement de l'intégration en émettant et recevant des trames de tous les types.

Pour réaliser ce test représenté à la figure 4.23, nous avons intégré les deux sous-niveaux sur chaque microprocesseur, ce qui permet de converser d'un niveau paquet vers l'autre.

Le fonctionnement de cette intégration qui constitue la version finale du niveau trame est commenté au chapitre 5.2.

Signalons néanmoins qu'on peut aller chercher les versions exécutables des programmes dans la directory/ ARTLAMB/ INTEGR/ EXEINT sous les noms TRETTD pour la version adaptée au microprocesseur ETTD et TRETCD pour la version écrite pour le microprocesseur ETCD.

4.3. Tests de fonctionnement

Comme nous l'avons noté dans le paragraphe consacré à l'introduction, les tests de fonctionnement sont exclusivement des tests du hardware. Ils seront exécutés quand un programme qui a déjà tourné correctement, ne produit plus les résultats désirés.

Nous nous intéresserons dans ce paragraphe aux différents tests hardwares à effectuer sur le microprocesseur quand le niveau trame ne fonctionne plus normalement.

En fait, une partie de ces tests a déjà été présentée au cours de ce chapitre. On peut en effet considérer la partie consacrée aux tests préliminaires comme un ensemble de tests de fonctionnement.

Le premier élément pouvant être vérifié est le fonctionnement du X21 bis. Si lors de l'exécution des programmes l'allumage n'est pas conforme aux règles présentées dans le mémoire de Messieurs Zone et Beudin, on peut considérer qu'un mauvais fonctionnement s'est produit lors de l'initialisation de la jonction physique.

Le deuxième test de fonctionnement pouvant être réalisé est celui du HDLC. Si les transferts ne s'opèrent pas correctement, l'erreur peut provenir d'un mauvais fonctionnement du HDLC. Celui-ci peut être testé avec les différents programmes expliqués au paragraphe 4.2.2.1.2.

Une troisième vérification peut être celle du fonctionnement correct du DMA. Si les transferts ne sont pas adéquats et que les tests du HDLC sont positifs, le bon fonctionnement du DMA peut alors être mis en cause. Il peut être contrôlé par l'utilisation des programmes présentés au paragraphe 4.2.2.1.3.

Un autre élément hardware pouvant provoquer des résultats inattendus est la mémoire. Le mauvais fonctionnement d'un "chip" peut éventuellement être la cause d'erreurs. On peut aisément vérifier le bon comportement d'une mémoire "suspecte" en inversant les deux cartes ou en allant examiner si les valeurs de ses cases sont celles attendues.

Le lecteur pourra trouver en annexe D quelques programmes de tests utilisés pour la mise au point du sous-niveau trame 1.

Chapitre 5 - Utilisation pratique du niveau trame

5.1. Mode d'emploi des microprocesseurs

a) Architecture

Avant de décrire le fonctionnement des microprocesseurs, nous allons présenter leur architecture schématisée à la figure 5.1 (*).

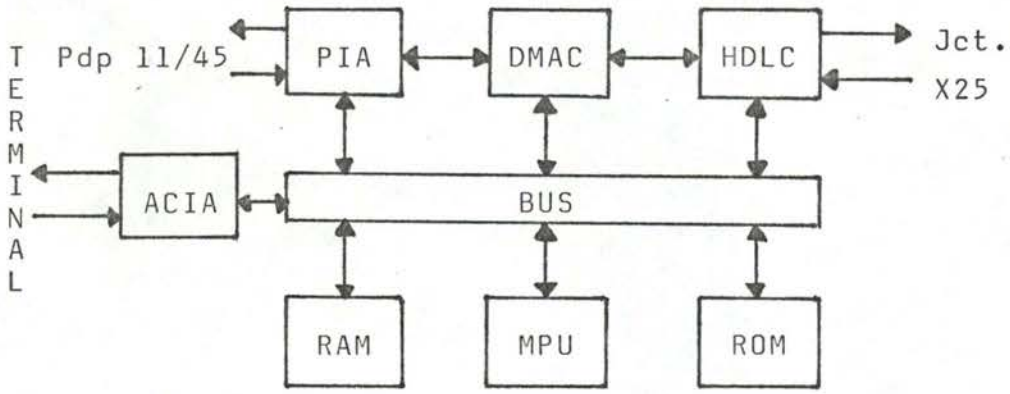


Fig. 5.1 - Architecture d'un microprocesseur

Les différents éléments qui les composent sont les suivants :

- un microprocesseur Motorola 6800 (MPU) travaillant sur un mot de 8 bits et possédant un bus données bi-directionnel et un bus adresses de 16 bits;
- une mémoire à lecture seule (ROM)
- une mémoire lecture-écriture (RAM)
- un interface pour périphérique (PIA) qui est un interface entre le microprocesseur et des périphériques. Il permet d'accéder aux périphériques par deux bus bi-directionnels de données;

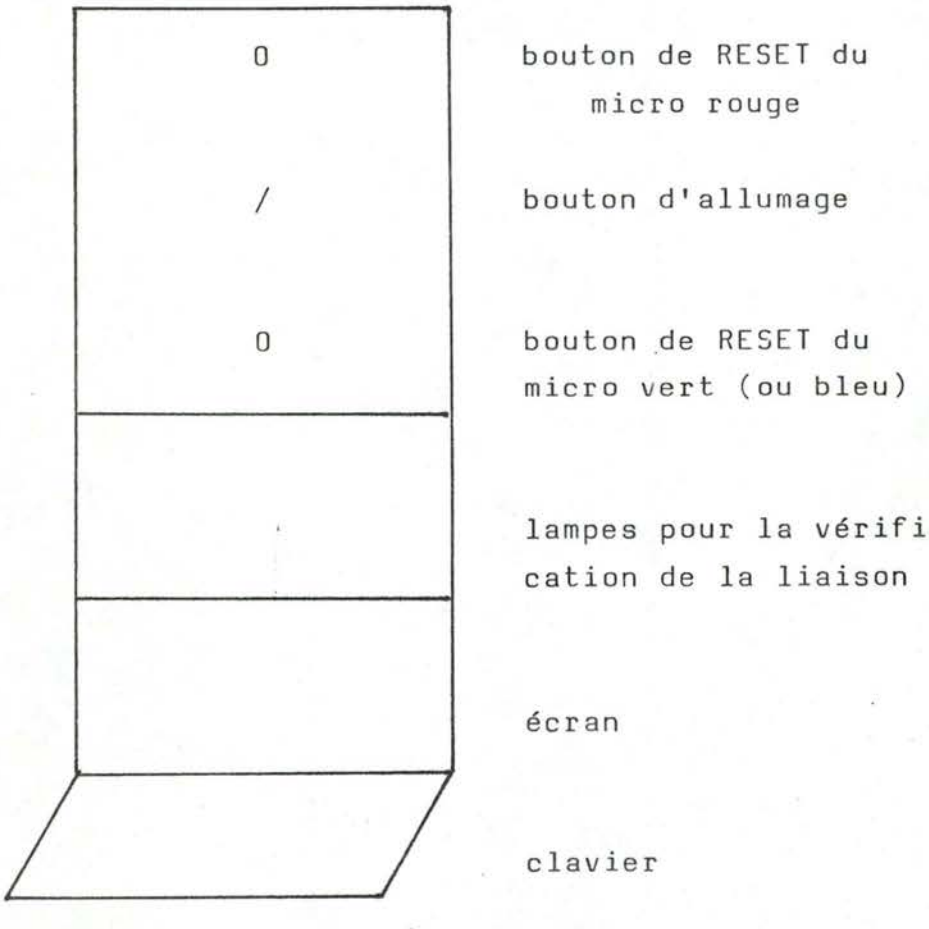
(*) Ce schéma a été repris du mémoire de J. Art.

- un interface pour communications asynchrone (ACIA) qui est un interface entre le microprocesseur et un périphérique travaillant en mode série asynchrone;
- un interface HDLC équipé d'un circuit intégré spécialisé réalisant toutes les fonctions du sous-niveau trame 1;
- un contrôleur d'accès direct mémoire (DMAC) qui accélère les opérations d'entrées-sorties en contrôlant les PIA'S et l'interface HDLC.

b) Description

Le schéma de la disposition des microprocesseurs dans le local de Louvain-la-Neuve est présenté à la figure 5.2.

Fig. 5.2 - Disposition des microprocesseurs



c) Caractéristiques

Le microprocesseur rouge est l'ETCD. Son nom est /DEV/DR1. Il possède 16 K-bytes de mémoire à partir de l'adresse 0000.

Le microprocesseur vert est l'ETTD. Son nom est /DEV/DR2. Il possède 16 K-bytes de mémoire à partir de l'adresse 0000.

Chacun d'eux possède également 1 K-bytes à partir de D000 et 128 octets à partir de DF00 (pour le moniteur).

d) Raccordement

Le raccordement d'un microprocesseur au terminal peut s'effectuer comme suit :

- s'assurer que le cable bleu (raccord à la pdp 11/05 derrière les microprocesseurs est déconnecté,
- connecter soit le microprocesseur ETCD avec la prise munie d'une étiquette rouge, soit le microprocesseur ETTD avec la prise munie d'une étiquette verte,
- placer le bouton d'allumage vers le bas pour le mettre dans l'état ouvert.

e) Commandes

En poussant sur le reset d'un microprocesseur, on reçoit le caractère ">" qui signale qu'on peut envoyer une commande au moniteur.

Les commandes sont les suivantes :

R — permet d'obtenir le contenu des registres, à savoir, dans l'ordre : CC REG, REG B, REG A, INDEX REG, P C, STACK POINTER.

M,aaa — permet d'obtenir la valeur, en hexadécimal, du contenu de la case mémoire d'adresse aaaa.

M,aaaa/xx — permet de remplacer le contenu de la case mémoire d'adresse aaaa par xx.

M,aaaa suivi de 2 blancs
après la commande M,aaaa chaque fois que l'on pousse sur RETURN, on obtient les contenus des cases mémoires des adresses suivantes. On peut interrompre la séquence en tapant 2 blancs après la commande.

P,aaaa, bbbb — permet d'obtenir un dump de la mémoire à partir de aaaa jusqu'à bbbb sous forme formattee.
S1 = début de ligne
S9 = fin de fichier
le byte suivant indique le nombre d'octets sur la ligne
les 2 bytes suivants indiquent l'adresse
le dernier byte est un calcul de vérification.

L — équivaut à l'inverse de P. Il attend des données du terminal sous le même format.

G,aaaa — permet d'exécuter un programme à partir de l'adresse aaaa.

F — permet de transférer des programmes dans les microprocesseurs à partir de la pdp 11/45. Les fichiers dont le nom se termine par ".mas" sont en assembleur motorola. Ils peuvent être assemblés par la commande MAS. Le résultat de l'assemblage, qui se trouve dans le fichier m.out, est exécutable par le microprocesseur. Il peut y être transférer en donnant au moniteur du micro la commande "F" et en donnant à UNIX la commande
cp m.out /DEV/DR1 ou
cp m.out /DEV/DR2.

5.2. Mise en oeuvre du niveau trame

La figure 5.3 présente la structure des directories du niveau trame. Nous trouverons ci-dessous les commentaires quant à leur contenu.

ARTLAMB

TRAME 1 : programmes des tests du sous-niveau trame 1

PRELIM : programmes des tests préliminaires : X21 bis, HDLC, DMA

TRA 1 : programmes des tests du sous-niveau trame 1 proprement dit : émission, réception

TRAME 2 : programmes du sous-niveau trame 2

TRA 268 : programmes de SIMTRA pour les microprocesseurs

TRA 210 : programmes de SIMTRA pour la pdp 11/10

INTEGR : programmes de l'intégration des 2 sous-niveaux : niveau trame

ART : programmes réalisés par J. Art

SIM68 : programmes simulant le niveau paquet : OPEN, CLOSE, GET, PUT.

Les directories commençant par "SOU" contiennent des versions sources tandis que celles débutant par "EXE" contiennent des versions exécutables.

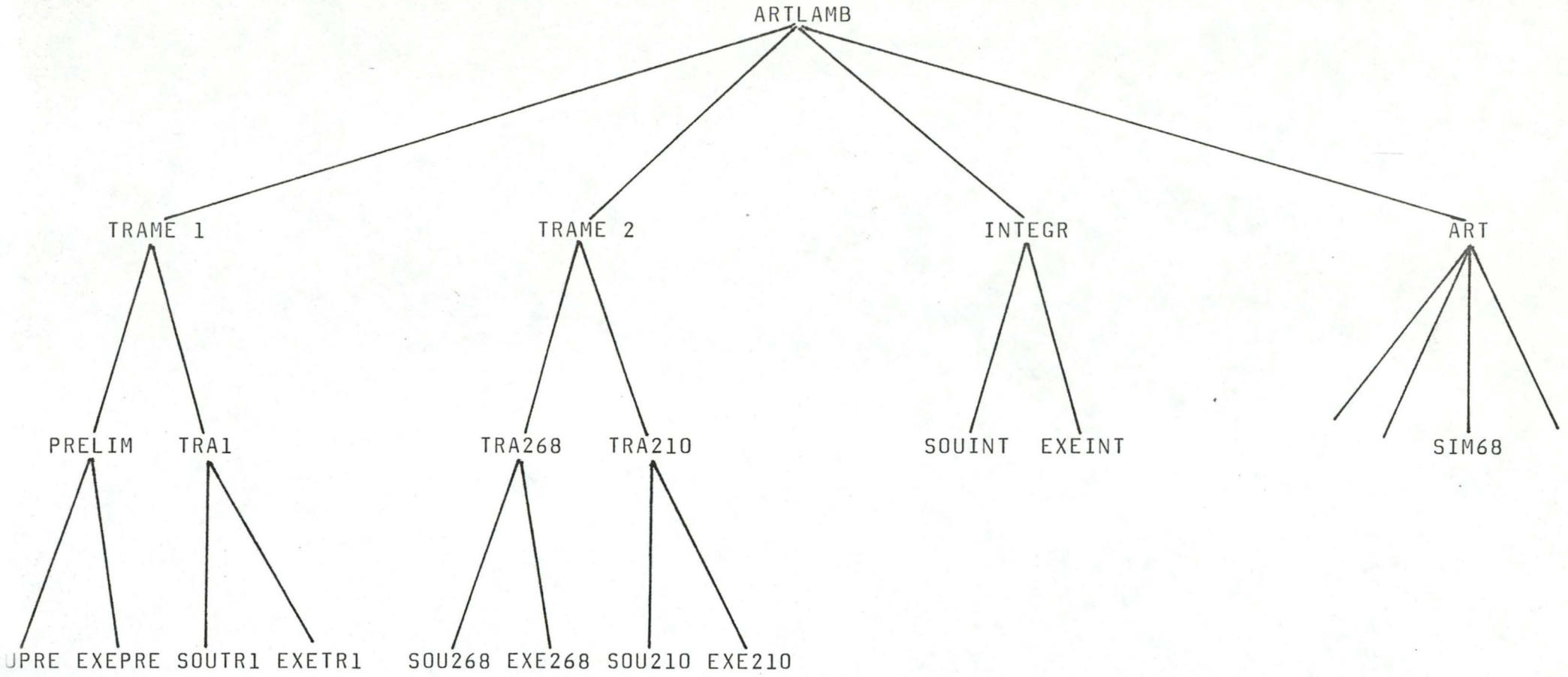


Fig. 5.3 - Structure des directories du niveau trame

Il est à noter que les programmes du sous-niveau trame 1 sont insérés en annexe E. Quand il s'agit d'une modification d'un programme de SIMTRA, les deux versions sont annexées afin de faire clairement apparaître les changements.

Nous avons tout d'abord le programme TRAME1.MAS qui contient tous les modules d'émission et de réception du HDLC et du DMA.

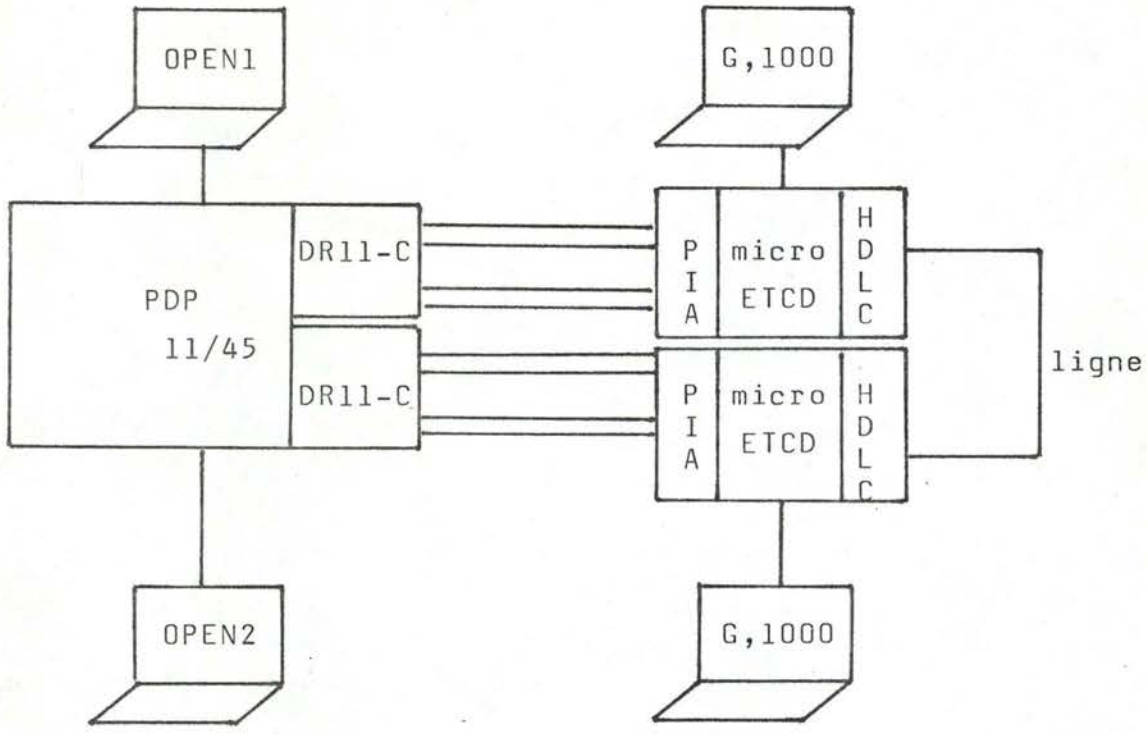
Suivent les programmes qui définissent les variables et buffers utilisés par le niveau trame. Apar est la version incluse dans SIMTRA tandis que Apartd.mas et Apared.mas sont les programmes qui font partie de l'intégration, respectivement pour l'ETTD et l'ETCD.

Le programme INIT68 réalise toutes les initialisations à entreprendre pour SIMTRA tandis que les versions modifiées INITCD et INITTD.MAS les effectuent pour l'intégration. Celle-ci oblige en effet à initialiser l'HDLC et le DMA, les nouveaux outils de transfert.

Atrame, enfin, est le programme de SIMTRA qui contient la boucle d'interruption. Atrame2 est sa nouvelle version : dans la boucle sont incluses les interruptions provoquées par le HDLC et le DMA.

La configuration utilisée pour mettre en oeuvre le niveau trame est présentée à la figure 5.4. Les deux versions intégrées tournent dans leur microprocesseur respectif et les niveaux paquets conversent entre eux grâce à deux terminaux connectés à UNIX.

Fig. 5.4 - Configuration du niveau trame



Pour aller chercher les programmes à envoyer dans les microprocesseurs, on se positionne dans la directory/ ARTLAMB/ INTEGR/ EXEINT.

On transfert les programmes dans les microprocesseurs en faisant cp TRETCD/ DEV/ DR1 et cp TRETTD/ DEV/ DR2 et on lance les deux programmes au moyen de la commande G,1000 sur les terminaux connectés aux microprocesseurs.

Sur les terminaux reliés à la pdp 11/45, on fait/ ARTLAMB/ ART/ SIM68 pour aller chercher les programmes qui simulent le niveau paquet.

On lance alors un GET sur chacun des terminaux et la configuration est prête pour la liaison. On peut alors lancer une commande (OPEN, CLOSE) de chaque niveau paquet afin d'initialiser les buffers.

Le niveau paquet qui veut émettre un paquet vers l'autre côté exécute un PUT et émet ensuite l'information à transférer.

Afin de compléter le dialogue, des messages accompagnent les transferts pour signaler comment s'exécutent les échanges. Les messages sont des mots de 16 bits dont les significations sont détaillées aux pages 68 à 71 du mémoire de monsieur Art.

Chapitre 6 - Conclusion

La version actuelle du niveau trame de la liaison X25 est correcte et réalise effectivement les transferts des paquets.

Le sous-niveau trame 2 réalisant la gestion de l'automate a été mis au point l'année passée par monsieur Art.

Le sous-niveau trame 1 comprenant les routines de gestion de l'interface HDLC et du contrôleur DMA ont été écrites et testées cette année, de même que l'intégration des deux sous-niveaux.

De plus, des programmes de tests permettent de vérifier le bon fonctionnement du sous-niveau trame 1.

Les difficultés rencontrées ont été de deux ordres.

D'une part, le hardware a présenté quelques défauts dans le courant de cette année. Il nous a obligés à plusieurs corrections et a ainsi retardé les tests du software.

D'autre part, les outils du microprocesseur pour la mise au point de programmes n'étant pas très développé, cela a ralenti les tests des programmes de trame 1.

Plusieurs projets me paraissent intéressants dans le cadre de l'évolution de la liaison.

Tout d'abord, améliorer la fiabilité des microprocesseurs afin de diminuer les risques de panne hardware.

Ensuite, intégrer les niveaux trame et paquet afin de tester le projet initial.

Enfin, effectuer des tests de performances en fonction de paramètres tels que la longueur des buffers afin d'aboutir à une version optimale du niveau trame.

ANNEXE A :

Programmation du HDLC

ESTERN DIGITAL
OS/LSI

SD1933A/B

SYNCHRONOUS DATA LINK CONTROLLER

FEATURES

- SYNCHRONOUS FULL DUPLEX OPERATION
- NRZI ENCODE/DECODE OPTION
- DC TO 1.5M BITS/SEC BAUD RATE
- SDLC PROTOCOL INCLUDING ZERO INSERTION AND DELETION, CRC GENERATION AND CHECK, ABORT, FLAG AND INVALID Frame detect
- IDLE detect
- Full set of modem interface signals
- End of Data Block sensing
- Variable byte length (5, 6, 7 or 8 bits)
- Residual byte capability
- DIRECT ACCESS MODE FOR DMA APPLICATIONS
- SYSTEM COMPATIBILITY
- Straight-forward Computer Interrupts
- Double Buffering of Data
- TTL compatible
- PROGRAMMABLE MODEM CONTROL INTERRUPTS
- ERROR DETECTION
- CRC, Underrun, Overrun, Aborted and Invalid frame errors
- GO AHEAD OPTION FOR LOOP APPLICATIONS

PPLICATIONS

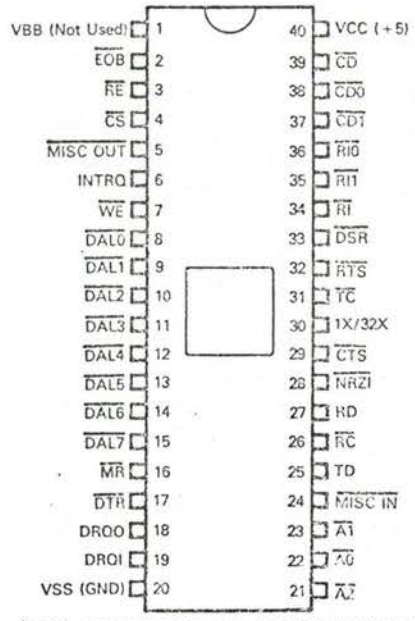
- AINFRAME COMMUNICATIONS PARTS
- ERMINAL COMMUNICATIONS PARTS
- ELECOMMUNICATIONS SWITCHING NETWORKS
- ACKET SWITCHING

ENERAL DESCRIPTION

e SD1933 is a MOS/LSI device which performs the functns of interfacing a parallel digital system to a synchrons serial data communications channel employing SDLC e protocol. The SD1933 is compatible with the IBM DLC General Information Specification, the ANSI and e ADCCP X3S34/589 specifications.

s fabricated in N-channel depletion load MOS technology c is TTL compatible on all inputs and outputs.

PIN CONNECTIONS



(PIN POSITIONS TO BE ASSIGNED)

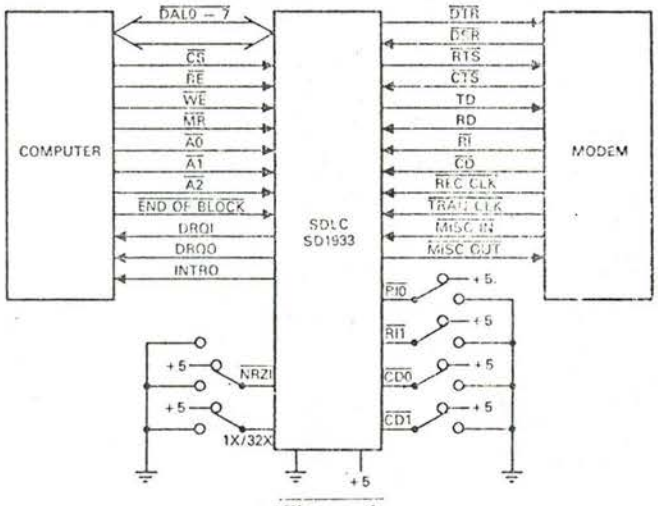
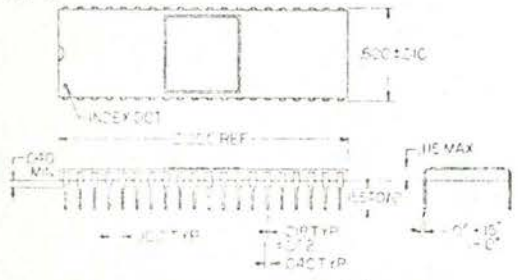


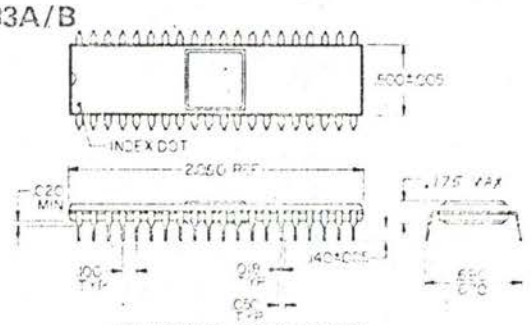
Figure 1

D1933A/B



CERAMIC PACKAGE

SD1933A/B



PLASTIC PACKAGE

ERN DIGITAL
P O R A T I O N

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

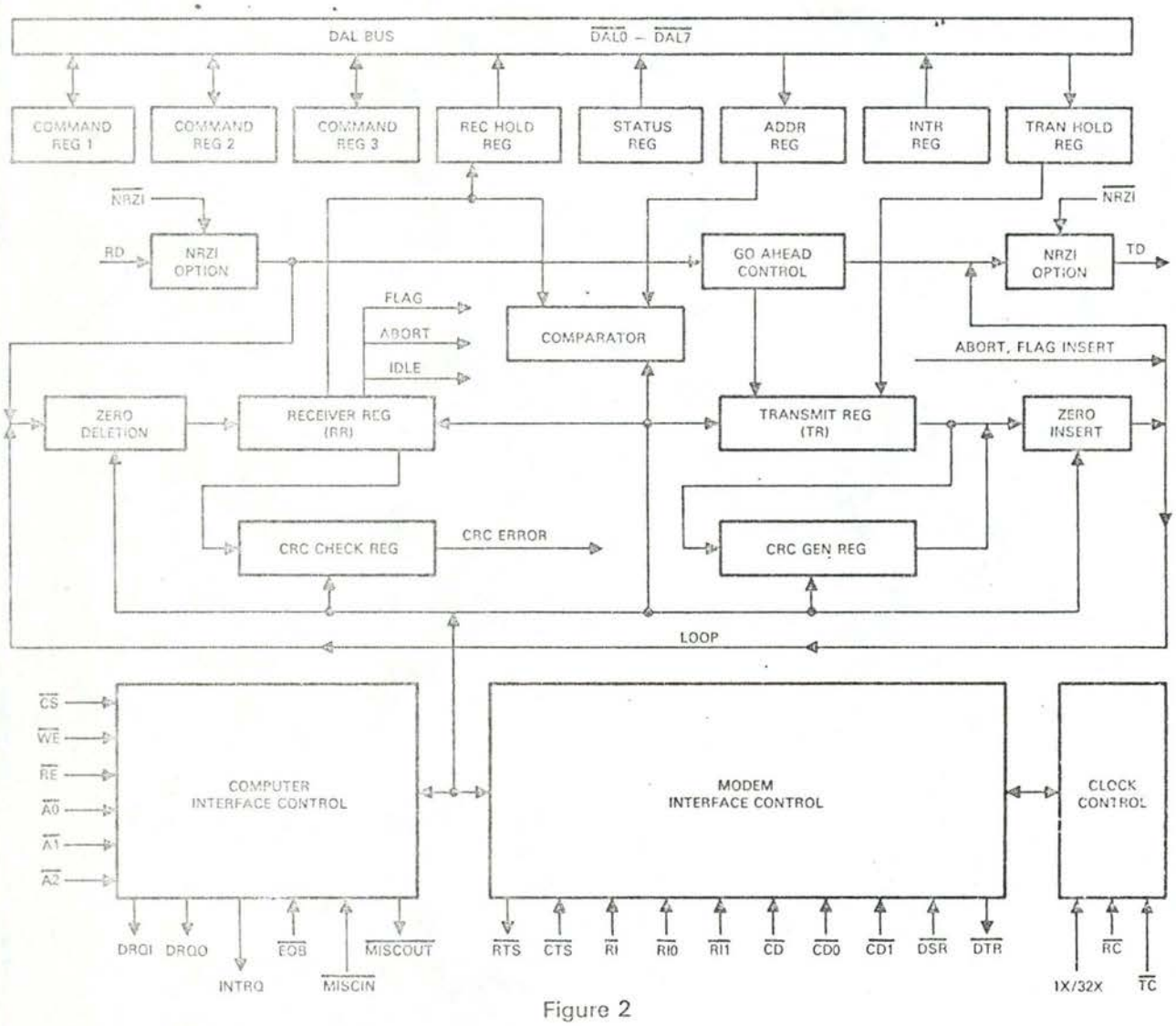


Figure 2

RGANIZATION

he SDLC block diagram is illustrated above and scribed below:

ceiver Register — This 8-bit register inputs the ceived data at a clock rate determined by the ceiver clock. The incoming data is assembled to a 6, 7 or 8 bit character length and then transferred the Receiver Holding Register (RHR). At this time, ta Request Input (DRQI) is made active informing e computer that the RHR contains data.

ceiver Holding Register — This 8-bit parallel ister presents assembled receiver characters to e DAL bus lines when activated via a read opera- n. When the RHR is read by the computer, DRQI made inactive.

Comparator — This 8-bit comparator is used to compare the contents of the Address Register with the address field of the incoming frame. This feature is enabled by a bit in the Command Register. If enabled and there is a match, the received frame is inputted and DRQI's are generated. If enabled and there is not a match, the received frame is discarded. If not enabled, all received frames are inputted to the computer.

Transmitter Holding Register — This 8-bit register is loaded with data from the DAL lines by a write operation. DRQO is also reset by the write operation. The Data is transferred to the Transmitter Register when the transmitter section is enabled and the Transmitter Register is ready for new data. During this transfer Data Request Output, (DRQO) is made active informing the computer that the THR is again empty.

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

Transmitter Register — This 8-bit Register is loaded from the Transmitter Holding Register and is serially shifted out to the Transmit Data output. An ABORT or FLAG may be loaded into this register under program control.

Command Registers — These 8-bit registers are loaded from the $\overline{DAL}$ lines by a write operation or read on the $\overline{DAL}$ lines by a read operation. The registers contain all the SDLC commands.

Status Register — This 8-bit register is read onto the $\overline{DAL}$ lines by a read operation. This register contains the overall status of the SD1933.

Address Register — This 8-bit register is loaded from the $\overline{DAL}$ lines by a write operation. It contains the address which is compared to the address byte of a frame (when the address compare feature is selected).

Interrupt Register — This 8-bit register is read onto the $\overline{DAL}$ lines by a read operation. This register contains the cause of the current interrupt request.

Data Access Lines — The $\overline{DAL}$ is an 8-bit bi-directional bus.

Zero Deletion — The received data stream is continuously monitored. Upon receiving five contiguous one bits, the sixth bit is inspected. If the sixth bit is a zero, it is automatically deleted from the data stream. If the sixth bit is a one the seventh bit is inspected; if it is a zero, a FLAG sequence has been received; if it is a one an ABORT sequence has been received.

Zero Insertion — A zero bit is automatically inserted following five contiguous one bits anywhere between the beginning FLAG and the ending FLAG of a frame. The insertion of the zero bit thus applies to the contents of the Address, Control, Info and FCS field (including the last five bits of the FCS).

CRC Generation and Check — The SDLC contains a 16-bit CRC check register and a 16-bit CRC generation register.

The generator polynomial is:

$$G(X) = X^{16} + X^{12} + X^5 + 1$$

The transmitter and receiver initialize the remainder value to all ones before CRC accumulation starts. The polynomial is multiplied by X^{16} and is divided by G(X). Inserted 0's are not included in the accumulation. Under program control, the complement called the frame check sequence (FCS) is sent with high order bit first.

Go-Ahead Option — When operating as a secondary station in the Go Ahead mode (loop configuration) the SDLC chip operates as a data repeater with a one-bit delay between Receiver and Transmitter. The SDLC will suspend the repeater function and initiate transmission when a Go Ahead pattern is detected (zero followed by 7 one's) and the transmitter section is enabled.

1X/32X Option — When high the chip expects 1X clocks (both receiver and transmitter) and will send data accordingly. When low the chip expects 32X clocks and will internally generate 1X clocks. The receiver syncs its 1X clock to each data transition. Note that transmitted data changes on the rising edge of $\overline{\text{Transmit Clock}}$ and received data is sampled on the falling edges of $\overline{\text{Receive Clock}}$.

NRZI Option — When this option is chosen the transmitter encodes the transmitted data to the NRZI format and the receiver expects NRZI data and thus decodes the NRZI encoding. In NRZI encoding the output remains in the same state to send a binary 1 and changes state to send a binary 0.

End of Block Option — This will cause the FCS command (type 3) to be executed as soon as the present command is completed. This option is designed to be used when blocks of data are being sent during a frame.

CONTROLLING AND MONITORING THE SDLC

Controlling the SDLC is done with the command registers. Monitoring is done by use of the Status and Interrupt Registers. The Command Registers dictate what the transmitter will send: the type of information (ABORT, FLAG, FCS or DATA), the number of bits per byte, and the number of bits in the residual byte. Similarly they tell the receiver the types of frames to look for: the number of bits per byte, whether to perform an address compare and whether to watch for an extended address.

The Status and Interrupt Registers perform the monitoring function. The Status Register indicates if an Aborted or Invalid frame has been received, if an Idle was received, etc. The Interrupt Register indicates if End of Messages have been received, if the transmission of a frame is complete, plus it monitors the states of INTRQ, DRQI, and DRQO.

The Command Registers also control Data Terminal Ready (DTR), Misc. Out, and the activation of both the Transmitter and Receiver.

COMMAND REGISTER FORMATS

The format of Command Register 1 (CR1) is shown below followed by a description of each bit location.

CR1

17	16	15	14	13	12	11	10
ACT REC	ACT TRAN	TC1	TC0	TBL1	TBL0	DTR	Misc Out

Activate Receiver (CR17) — This bit when set activates the receiver which begins searching for frames.

Activate Transmitter (CR16) — This bit, when set, activates the transmitter, sets RTS, and when CTS is received begins executing the transmitter commands; if in the Go-Ahead Mode, the transmitter waits for a Go-Ahead (zero followed by 7 ones) before executing the command.

Transmitter Commands (CR15, 14) — These bits indicate to the transmitter what data to send; either a FLAG, an ABORT, a Frame Check Sequence (FCS), or Data in the THR. Refer to Transmitter Commands section for description.

Transmitter Byte Length (CR13, 12) — These bits designate the number of bits there will be in each Data byte. Each data byte may be 5, 6, 7 or 8 bits long.

TBL1	TBL0	Bits Per Byte
0	0	8
0	1	7
1	0	6
1	1	5

DTR Command (CR11) — This bit controls the Data Terminal Ready (DTR) signal to the data set. When CR11 is a zero, DTR is off. When CR11 is a one, DTR is on. When the Self-Test mode is selected DTR is forced to a zero.

Miscellaneous Output (CR10) — This bit controls the Miscellaneous Output to the data set. When CR10 is a zero Misc. Out is off and when it is a one Misc. Out is on.

The format of Command Register 2 (CR2) is shown below followed by a description of each bit location.

NOTE: CR2 should be loaded only when both the Transmitter and Receiver are not enabled.

CR2

27	26	25	24	23	22	21	20
Control Bytes	ADDR Comp	EXT ADDR	RBL1	RBL0	GA	Self Test	Auto Flag

Number of Control Bytes (CR17) — When set, this bit tells the SDLC to expect two control bytes per frame. When zero, the SDLC expects one control byte.

Address Compare Enable (CR26) — This bit, when set, causes the receiver to inspect the first incoming address byte. If there is: 1) a match with the Address Register or 2) if Extended Address is not selected and the address is all ones or 3) if Extended Address is selected and all but the 2^7 bit are ones, the rest of the frame is inputted. Otherwise the receiver searches for a new frame. If not set, all frames are inputted.

Extended Address Enable (CR25) — This bit, when set, will cause the receiver to input another address byte if the 2^7 bit in the current address byte is a one.

Receiver Byte Length (CR24, 23) — These bits indicate to the receiver how many bits per byte it should assemble for the I-field. The I-field bytes may be 5, 6, 7 or 8 bits long.

RBL1	RBL0	Bits Per Byte
0	0	8
0	1	7
1	0	6
1	1	5

Go-Ahead (CR22) — This bit, when set causes the chip to work in the Go-Ahead mode, as used in a loop type configuration.

Self-Test Mode (CR21) — This bit when set deactivates DTR and loops the Transmitter Data output around to the Receiver Data input.

Auto Flag (CR20) — If: 1) the Auto Flag bit is set, 2) the transmitter is not in a frame, and 3) the Data Command is executed, the Transmitter will send FLAGS before sending out the contents of the THR.

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910 595-1139

This eliminates the need to execute the FLAG Command at the beginning of a frame.

The format of Command Register 3 (CR3) is shown below followed by a description of each bit.

CR3

37	36	35	34	33	32	31	30
unused	unused	unused	unused	unused	RBL2	RBL1	RBL0

Residual Byte Length (CR32, 31, 30) — These bits determine what length the residual byte will be. If no residual byte is to be sent, they must be set to zero.

Res2	Res1	Res0	Residual Byte Length
0	0	0	no residual byte sent
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

STATUS REGISTER FORMAT

The format of the Status Register is shown below followed by a description of each bit.

7	6	5	4	3	2	1	0
RI	CD	DSR	MISC IN	RECEIVE IDLE	AF/or IF/RES2	0RUN/ RES1	CRC/ RES0

Ring Indicator (SR7) — This bit is a copy of the Ring Indicator input to the SDLC.

Carrier Detect (SR6) — This bit is a copy of the Carrier Detect input to the SDLC.

Data Set Ready (SR5) — This bit is a copy of the Data Set Ready input to the SDLC.

Miscellaneous Input (SR4) — This monitors the level of the Miscellaneous Input Pin.

Receive IDLE (SR3) — This bit is set when 2 ABORT's in a row (14 ones) have been received indicating that the terminal which is sending data is in an IDLE state.

Receive Error Bits/Residual Count (SR2-SR0) — If an REOM without Errors is received these bits indicate the number of residual bits which were received. If an REOM with Errors is received, then SR2-SR0 indicate which errors occurred.

	SR2	SR1	SR0
REOM w/o Errors	RES2	RES1	RES0
REOM w/Errors	Aborted frame or Invalid (less than 32 bits) frame	Overrun Error (DRQI not serviced)	CRC Error

INTERRUPT REGISTER FORMAT

An Interrupt is generated when INTRQ=1. This signal remains true until the Interrupt Register is read at which time INTRQ=0 and the interrupt is serviced. A new interrupt may occur if one was pending. The interrupt register is shown below followed by a description of each bit.

IR

7	6	5	4	3	2	1	0
REOM no Err	REOM Errors	Xmit Opcom w/no errors	Xmit Opcom w/ under-run	DSC	DRQI	DRQ0	INTRQ

Received End of Message — No Errors (IR7) — This bit is set when an End of Message has been detected with no errors.

Received End of Message — Errors (IR6) — This bit is set when an End of Message has been detected with errors. Errors include CRC, Overrun, Invalid Frame, and Aborted Frame as determined by the Status Register.

XMIT Operation Complete — No Errors (IR5) — This bit is set when the command in CR1 has been completed and there were no errors.

XMIT Operation Complete with Underrun Error (IR4) — This bit is set when the indicated command in CR1 has been completed and there was an Underrun error. An Underrun error occurs when the THR was not loaded by the computer in time.

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

Data Set Change (IR3) — This bit is set whenever: 1) Carrier Detect (CD) changes state (if so programmed) 2) DSR changes state 3) Ring Indicator (RI) changes state (if so programmed).

The programming of CD and RI is done via CD1-CD0 and RI1-RI0 as follows:

CD1 RI1	CD0 RI0	Interrupting Edge
0	0	Rising & Falling
0	1	Rising
1	0	Falling
1	1	Neither

DRQI (IR2) — This is an output pin but is included as Bit 2 in the IR as an added convenience.

DRQ0 (IR1) — This is an output pin but is included as Bit 1 in the IR as an added convenience.

INTRQ (IR0) — This bit, when set, indicates an interrupt and that there are one or more bits set in bit positions 3 through 7 of the Interrupt Register. This bit is reset when the IR is read by the Computer.

Upon receipt of master reset, the IR is cleared. Bits 7-3 are reset whenever the IR is read by the computer and there is an interrupt request. Bits 7/6 and 5/4 are mutually exclusive. DRQI is serviced when the RHR is read by the computer. DRQO is serviced when the THR is loaded by the computer.

TRANSMITTER COMMANDS

CR

15	14		
TC1	TC0	Command	Action
0	0	Data	DRQO
0	1	ABORT	INTRQ
1	0	FLAG	INTRQ
1	1	Frame Check Sequence (FCS)	INTRQ

Data 00 (Type 0) — Upon receipt of this command the Transmitter Holding Register (THR) is transferred into the TR if the THR is loaded and TR is done shifting out any previous data. When the THR is transferred to the TR a DRQO is generated indicating that the THR is empty. If the THR has not been loaded with a new byte by the time the TR is shifted out, an INTRQ with the XMIT-Underrun Error bit set is generated and ABORT's without subsequent INTRQ's are sent.

When the DATA command is executed while not in a frame and the THR is not loaded, continuous FLAGs without INTRQs will be sent if the AUTO FLAG option is chosen, otherwise continuous ABORTs without INTRQs will be sent until the command is changed or the THR is loaded.

Abort 01 (Type 1) — Upon receipt of this command, an ABORT sequence (8 ones) is loaded into the TR and XMIT operation complete is generated (INTRQ = 1). After the interrupt has been serviced the command may change. If a new command has not been received by the time the last bit is out of the TR, another ABORT sequence is loaded into the TR and another interrupt generated. This sequence continues until the command is changed.

Flag 10 (Type 2) — Upon receipt of this command, a FLAG (01111110) is loaded into the TR and XMIT operation complete is generated (INTRQ = 1). After the interrupt has been serviced, the command may change. If a new command has not been received by the time the last bit is out of the TR, another FLAG is loaded into the TR and another interrupt is generated. This sequence continues until the command is changed.

Frame Check Sequence (FCS) (Type 3) — Upon receipt of this command, the Residual byte (which the chip automatically transfers into the THR) will be sent, provided RES2-RES0 ≠ 0. Following the Residual byte will come the FCS then a FLAG along with an INTRQ (XMIT operation complete) thus ending the frame. After the interrupt has been serviced, the command may change. If the FCS command is executed while not in a frame, and if AUTO FLAG is not chosen the transmitter will send ABORT's without INTRQ's. If AUTO FLAG is chosen, continuous FLAG's with INTRQ's will be sent.

RECEIVER OPERATION

When the Receiver is activated (CR17), it clocks data into the Receiver Register (RR) on the positive transitions of clocks and begins searching for a FLAG (01111110). When a FLAG is detected, the next 8-bit character is assembled into the receiver and

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

inspected. Also, the CRC check register is preset to ones. If this character is another FLAG or an ABORT (7 or more contiguous ones) it is discarded and the next character is brought into the shift register and inspected. If this character is not a FLAG or an ABORT, a beginning of a frame is detected. If CR26 is off (address compare not enabled) the character is loaded into the RHR, the DRQI is generated, and the frame is inputted. If CR26 is on, an address compare is performed. If there is a match, the frame is inputted. If there is no match, the frame is discarded and the receiver goes back to search for FLAG mode.

After receiving the Address byte(s), the Control byte(s) is inputted, then data characters are inputted in 5, 6, 7 or 8 bit bytes as indicated by RBL1-RBL0 (CR24-23). Note that if a DRQI is not serviced in time the Overrun Error bit is set at the end of the frame. Bytes are inputted until a FLAG or an ABORT ends the frame. At this time an INTRQ with one of the two Receive End of Message (REOM) bits is set — depending on if there were errors or not. Status Register bits SR0-SR2 indicate the errors, otherwise they indicate the number of bits in the Residual type.

After the end of the frame, the receiver begins searching for a new frame. Note that two frames may be separated by only one FLAG.

INPUT/OUTPUT OPERATIONS

The reading and writing of registers is controlled by the following signals:

Chip Select ($\overline{CS}$), Write Enable ($\overline{WE}$),

Read Enable ($\overline{RE}$), and $\overline{A0}$, $\overline{A1}$, $\overline{A2}$.

$\overline{A0}$, $\overline{A1}$, and $\overline{A2}$ determine which registers are accessed as follows:

A2	A1	A0	Register	
			Read	Write
0	0	0	CR1	CR1
0	0	1	CR2	CR2
0	1	0	CR3	CR3
0	1	1	RHR	AR
1	0	0	IR	THR
1	0	1	SR	—

Write — A write operation is initiated by the placement of a 3-bit address on the A2-A0 lines. $\overline{CS}$ and $\overline{WE}$ must then be pulled low. The data on the DAL lines is locked into the accessed register when either $\overline{CS}$ or $\overline{WE}$ is brought high. Refer to page 8 for timing diagrams.

Read — A read operation is initiated by the placement of the appropriate register address on the A2-A0 lines. $\overline{CS}$ and $\overline{RE}$ must then be pulled low. Data will be valid typically 100 ns later. Refer to page 8 for timing diagrams.

Immediately after the Interrupt Register is read IR0 and IR3-IR7 are reset. Likewise, after reading the Status Register, SR0-SR3 are reset.

SAMPLE TRANSMISSION

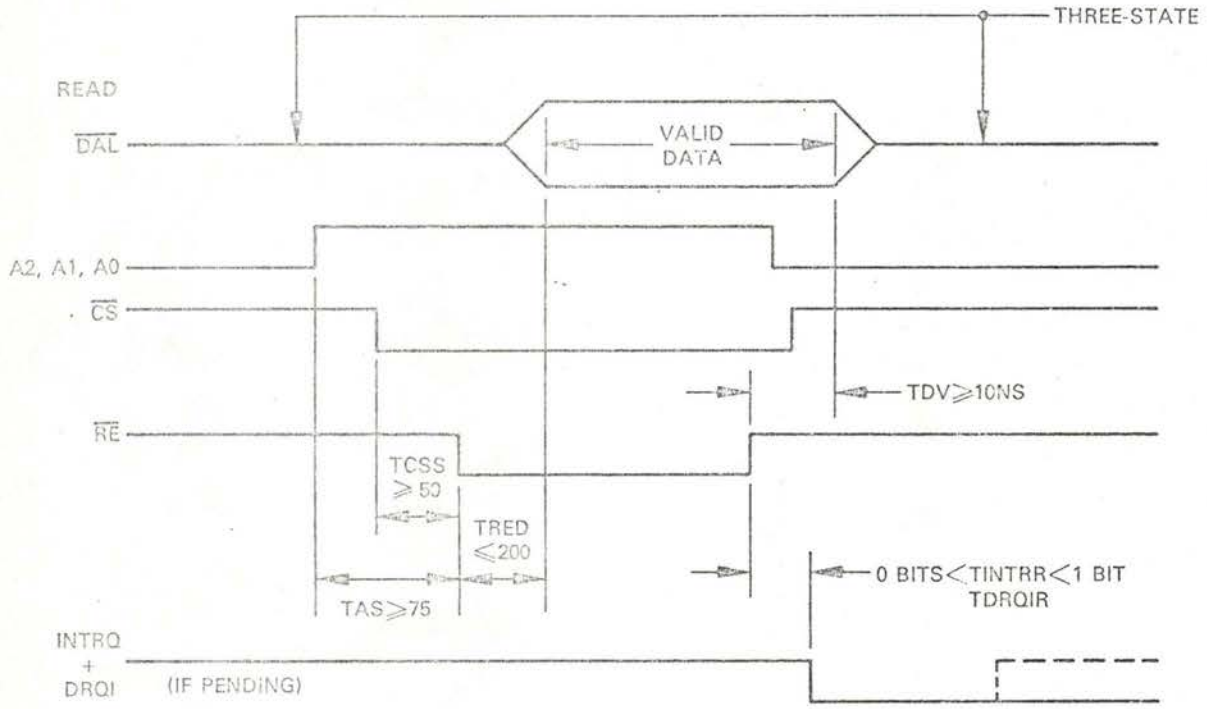
Below is a sample interchange between the SD1933 and the controlling computer with Auto Flag = 0.

Message to be sent:

FLAG	ADDRESS	CONTROL	I-FIELD	FCS	FLAG

Computer	Data	SD1933
ACT TRAN, TYPE 0→CR1	→	
	←	DRQO
TYPE 2→CR1	→	
	←	INTRQ*
READ IR, TYPE 0→CR1, ADDR→THR	→	
	←	DRQO
C→THR	→	
	←	DRQO
I→THR	→	
	←	DRQO
TYPE 3→CR1	←	
	←	INTRQ
READ IR, DEACT XMIT, TYPE 0→CR1	→	
*Unnecessary when Auto Flag = 1		

3128 RED HILL AVENUE, BOX 2180
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595 1139



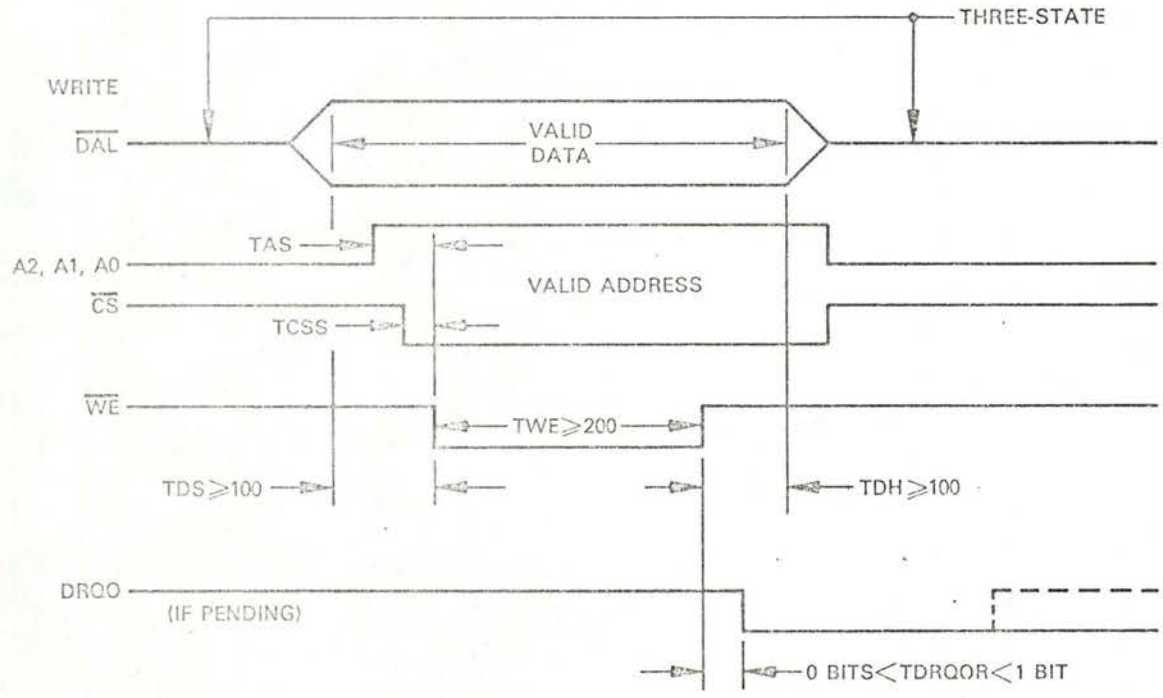


Figure 3

3128 RED HILL AVENUE, BOX 2180
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

Pin No.	Pin Name	Symbol	Function
	RECEIVE CLOCK	$\overline{RC}$	This input is the Receiver Data Rate Clock. It may be programmed to expect a 1X or 32X clock.
	1X/32X	1X/32X	When high the SD1933 expects a 1X clock; when low it expects a 32X clock.
	MISC. OUTPUT	$\overline{MISC\ OUT}$	This output is controlled by a bit in the control register and may be used as an extra programmable signal.
	MISC INPUT	$\overline{MISC\ IN}$	This input is read in the Status Register and may be used as an extra input.
	NRZI	$\overline{NRZI}$	This input, when low, causes the receiver to expect, and the transmitter to send NRZI encoded data.
	END OF BLOCK	$\overline{EOB}$	When low, the transmitter will execute the FCS command at the completion of the current command.
	WRITE ENABLE	$\overline{WE}$	This signal when low gates the contents of the DAL bus into the addressed register of a selected SDLC.
	INTERRUPT REQUEST	INTRQ	This output is made high whenever one of the following Interrupt Register bits is set: IR7-IR3.
	A2-0	$\overline{A2\text{-}0}$	A2-A0 are used to address the SDLC registers during read/write operations.
	REQUEST TO SEND	$\overline{RTS}$	This output is enabled by the control register and remains in a low state during data transmission from the SDLC.
	CLEAR TO SEND	$\overline{CTS}$	This input, when low, enables the transmitter section of the SDLC.
	DATA SET READY	$\overline{DSR}$	This input generates an interrupt when going On or Off. It appears as a bit in the status register.
	DATA TERMINAL READY	$\overline{DTR}$	This output is generated by a bit in the Control Register and indicates controller readiness.
	RING INDICATOR	$\overline{RI}$	This input from the Data Set may be programmed to cause a Data Set Change interrupt on the rising edge, falling edge, both edges or neither edge using the RI1-RI0 inputs.
	RI0-RI1	$\overline{RI0\text{-}RI1}$	Used to program the RING INDICATOR interrupts.
	CARRIER DETECT	$\overline{CD}$	This input from the Data set may be programmed to cause a Data Set Change interrupt on the rising edge, falling edge, both edges or neither edge using CD1-CD0 inputs.
	CD0-CD1	$\overline{CD0\text{-}CD1}$	Used to program the Carrier Detect interrupts.

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

Pin No.	Pin Name	Symbol	Function
	TRANSMIT CLOCK	$\overline{TC}$	This input is the Transmitter Data Rate Clock. It may be programmed to expect a 1X or a 32X clock.
	VCC	VCC	+5V
	VSS	VSS	Ground
	MASTER RESET	$\overline{MR}$	The control and status registers and other controls are cleared when this input is low.
	DATA ACCESS LINES	$\overline{DAL0\text{-}7}$	Eight bi-directional lines used for transfer of data, control, status and address information. Bit 7 is MSB.
	CHIP SELECT	$\overline{CS}$	When this signal is low it selects the chip for a read or write operation.
	READ ENABLE	$\overline{RE}$	This signal when low gates the contents of an addressed register from a selected SDLC onto the DAL.
	DATA REQUEST INPUT	DRQI	When high, DRQI indicates that the Receiver Holding Register contains valid data.
	DATA REQUEST OUTPUT	DRQO	DRQO, when high, indicates that the Transmitter Holding Register is empty.
	TRANSMITTED DATA	TD	This output is the transmitted serial data.
	RECEIVED DATA	RD	This input receives serial data.

ELECTRICAL CHARACTERISTICS

MAXIMUM RATINGS

Operating Temperature — 0°C to 70°C
Storage Temperature — −55°C to +125°C

OPERATING CHARACTERISTICS (DC)

T_A = 0°C to 70°C
V_{CC} = 50 ma Nominal
V_{SS} = 0V, V_{CC} = +5V ±.25V

Symbol	Characteristic	Min.	Typ.	Max.	Units	Conditions
I_{LI}	Input Leakage			10	μA	$V_{IN} = V_{cc}$
I_{LO}	Output Leakage			10	μA	$V_{OUT} = V_{cc}, V_{OUT} = V_{SS}$
V_{IH}	Input High Voltage	2.4			V	
V_{IL}	Input Low Voltage (All Inputs)			0.8	V	
V_{OH}	Output High Voltage	2.8			V	$I_O = -100\ \mu A$
V_{OL}	Output Low Voltage			0.40	V	$I_O = 1.6$ mA

3128 RED HILL AVENUE, BOX 2180
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

SDLC PROTOCOL

The basic unit of information transfer in SDLC is the frame which is defined below:

FLAG	ADDRESS	CONTROL	INFO FIELD	FRAME CHECK SEQUENCE	FLAG

Where:

FLAG = 01111110

Address field — one or more 8 bit bytes defining the particular station

Control field — one or two 8 bit bytes

Information field — Any number of bits (may be zero bits)

Frame Check Sequence — 16 bit error checking field

A minimum frame is 32 bits long and consists of the following:

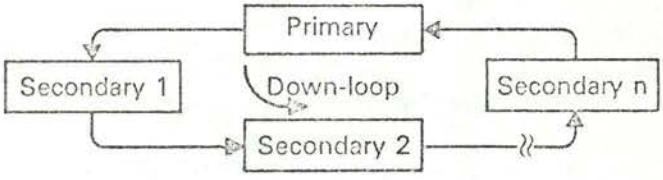
FLAG	ADDRESS	CONTROL	FCS	FLAG

The following features are also part of the SDLC protocol:

Zero Insertion/Zero Deletion — Within the 2 Flags of a frame, if there are more than 5 ones in a row, a zero is automatically inserted during transmission after the 5th one and it is deleted upon reception.

Frame Check Sequence (FCS) — During transmission a 16 bit cyclic redundancy check (CRC) calculation is performed on the ADDRESS byte(s), CONTROL byte(s) and the I-Field. This CRC is then transmitted after the I-Field and before the final FLAG. Upon reception the receiver also performs a CRC calculation on the incoming data. If there were no transmission errors the Receiver CRC equals zero.

LOOP APPLICATION (GO-AHEAD MODE) — The diagram below depicts a loop system.



Each secondary station in the loop is a repeater station. Transmissions from the Primary are relayed from station to station then back to the Primary. Any Secondary station finding its address in the ADDRESS field of a frame captures that frame for action. All received frames are relayed to the next station down the loop.

Whenever a Secondary station receives the GO-AHEAD pattern (7 ones) the Secondary may, at its option, suspend the repeater function and put its own transmission on the line. When it is finished it sends a GO-AHEAD pattern, then deactivates its transmitter and resumes the repeater function.

ABORT — In the non-LOOP mode frames may be ABORTed if desired simply by sending an ABORT (seven ones). Also 2 ABORTs in a row (14 consecutive ones) constitutes an IDLE.

3128 RED HILL AVENUE, BOX 2160
NEWPORT BEACH, CALIFORNIA 92663
(714) 557-3550 TWX 910-595-1139

ANNEXE B :

Programmation du DMA

MC6844

Advance Information

MC6

(N-Channel, Silicon-Gate)

DIRECT MEMORY ACCESS CONTROLLER (DMAC)

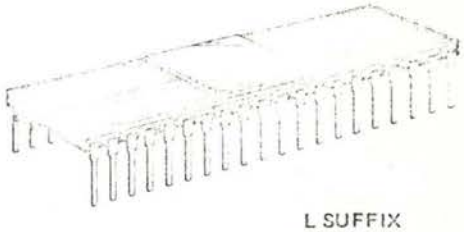
The MC6844 Direct Memory Access Controller (DMAC) performs the function of transferring data directly between memory and peripheral device controllers. It controls the address and data buses in place of the MPU in bus organized systems such as the M6800 Microprocessor System.

The bus interface of the MC6844 includes select, read/write, interrupt, transfer request/grant, and bus interface logic to allow the data transfer over an 8-bit bidirectional data bus. The functional configuration of the DMAC is programmed via the data bus. The internal structure provides for control and handling of four individual channels, each of which is separately configured. Programmable control registers provide control for the transfer location and length, individual channel control and transfer mode configuration, priority of servicing, data chaining, and interrupt control. Status and control lines provide control to the peripheral controllers.

The mode of transfer for each channel can be programmed as cycle-stealing or a burst transfer mode.

Typical applications would be with the Floppy Disk Controller (FDC) and the Advanced Data Link Controller (ADLC).

- Four DMA Channels, Each Having a 16-Bit Address Register and a 16-Bit Byte Count Register
- 1 M Byte/Sec Maximum Data Transfer Rate
- Selection of Fixed or Rotating Priority Service Control
- Separate Control Bits for Each Channel
- Data Chain Function
- Address Increment or Decrement Update
- Programmable Interrupts and DMA End to Peripheral Controllers



L SUFFIX
CERAMIC PACKAGE
CASE 715

P SUFFIX
PLASTIC PACKAGE
CASE 711

FIGURE 1 – M6800 MICROCOMPUTER FAMILY BLOCK DIAGRAM

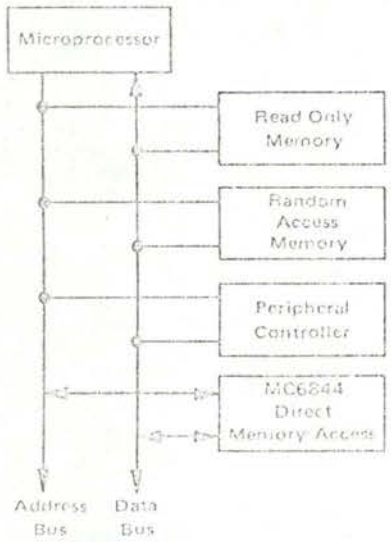
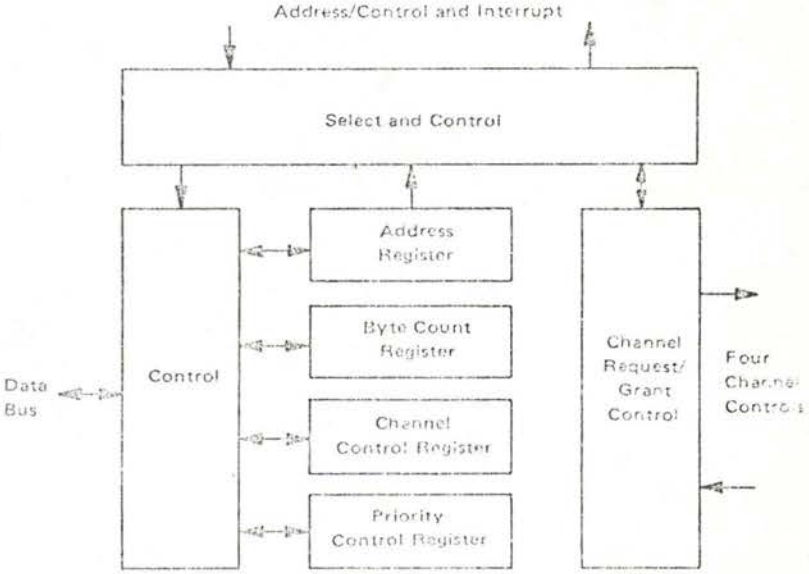


FIGURE 2 – DIRECT MEMORY ACCESS CONTROLLER BLOCK DIAGRAM



ADI-471
(Replaces MP-CC)

MAXIMUM RATINGS

Rating	Symbol	Value	Unit
Supply Voltage	V_{CC}*	-0.3 to +7.0	Vdc
Input Voltage	V_{in}*	-0.3 to +7.0	Vdc
Operating Temperature Range	T_A	0 to +70	°C
Storage Temperature Range	T_{stg}	-55 to +150	°C
Thermal Resistance	$R_{\theta JA}$	82.5	°C/W

* In respect to V_{SS}.

Permanent device damage may occur if ABSOLUTE MAXIMUM RATINGS are exceeded. Functional operation should be restricted to RECOMMENDED OPERATING CONDITIONS. Exposure to higher than recommended voltages for extended periods of time could affect device reliability.

RECOMMENDED OPERATING CONDITIONS

Rating	Symbol	Value	Unit
Power Supply Voltage	V_{CC}	+4.75 to +5.25	Vdc
Input Voltage	V_{IL} V_{IH}	-0.3 to +0.8 2.0 to V_{CC}	Vdc
Operating Ambient Temperature Range	T_A	0 to +70	°C

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V ±5%, V_{SS} = 0, T_A = -20 to +75°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
Input High Voltage		V_{IH}	V_{SS} +2.0	–	V_{CC}	Vdc
Input Low Voltage		V_{IL}	V_{SS} -0.3	–	V_{SS} +0.8	Vdc
Input Leakage Current (V_{in} = 0 to 5.25 V)	TxR00-3, φ2 DMA, $\overline{RES}$, DGRNT	I_{in}	–	–	2.5	μAdc
Three-State Leakage Current (V_{in} = 0.4 to 2.4 V)	A0-A15, R/$\overline{W}$ D0-D7	I_{TSI}	-10	–	10	μAdc
Output High Voltage		V_{OH}				Vdc
(I_{Load} = -205 μAdc)	D0-D7		V_{SS} +2.4	–	–	
(I_{Load} = -145 μAdc)	A0-15, R/$\overline{W}$		V_{SS} +2.4	–	–	
(I_{Load} = -100 μAdc)	All Others		V_{SS} +2.4	–	–	
Output Low Voltage (I_{Load} = 1.6 mAdc)		V_{OL}	–	–	V_{SS} +0.4	Vdc
Source Current (V_{in} = 0 Vdc, Figure 10)	$\overline{CS}$/Tx AKB	I_{CSS}	–	10	–	
Power Dissipation		P_D	–	500	–	mW
Capacitance		C_{in}				pF
(V_{in} = 0, T_A = 25°C, f = 1.0 MHz)	φ2 DMA		–	–	20	
	D0-D7, $\overline{CS}$, A0-A4, R/$\overline{W}$		–	–	12.5	
	All Others		–	–	10	
		C_{out}	–	–	12	pF

BUS TIMING CHARACTERISTICS (Load Condition Figure 11)

Characteristic		Symbol	Min	Max	Unit
READ TIMING (Figure 4)					
Address Setup Time	A0-A4, R/W, CS	t_{AS}	160	–	ns
Address Input Hold Time	A0-A4, R/W, CS	t_{AHI}	10	–	ns
Data Delay Time	D0-D7	t_{DDR}	–	320	ns
Data Access Time	D0-D7	t_{ACC}	–	480	ns
Date Output Hold Time	D0-D7	t_{DHR}	10	–	ns
WRITE TIMING (Figure 4)					
Address Setup Time	A0-A4, R/W, CS	t_{AS}	160	–	ns
Address Input Hold Time	A0-A4, R/W, CS	t_{AHI}	10	–	ns
Date Setup Time	D0-D7	t_{DSW}	195	–	ns
Data Input Hold Time	D0-D7	t_{DHW}	10	–	ns

CLOCK TIMING

Characteristic	Symbol	Min	Max	Unit
ϕ2 DMA (See Figure 4)				
Cycle Time	t_{cyc}	1000	–	ns
Pulse Width—High Low	PW_H PW_L	450 430	– –	ns
Rise and Fall Time	$t_{\phi r}, t_{\phi f}$	–	25	ns
DMA TIMING (Load Condition Figure 11)				
Tx RQ Setup Time (Figure 5) ϕ2 DMA Rising Edge ϕ2 DMA Falling Edge	 t_{TQS1} t_{TQS2}	 120 210	 – –	ns
Tx RQ Hold Time (Figure 5) ϕ2 DMA Rising Edge ϕ2 DMA Falling Edge	 t_{TOH1} t_{TQS2}	 20 20	 – –	ns
DGRNT Setup Time (Figure 6)	t_{DGS}	155	–	ns
DGRNT Hold Time (Figure 6)	t_{DGH}	10	–	ns
Address Output Delay Time (Figure 15) A0-A15, R/W, Tx STB	t_{AD}	–	270	ns
Address Output Hold Time (Figure 15) A0-15, R/W Tx STB	t_{AHO}	30 35	– –	ns
Address Three-State Delay Time (Figure 8) A0-A15, R/W	t_{ATSD}	–	700	ns
Address Three-State Recovery Time (Figure 8)	t_{ATSR}	–	400	ns
Delay Time (Figure 7) DROH, DROT	t_{DQD}	–	375	ns
Tx AK Delay Time ϕ2 DMA Rising Edge (Figure 7) DGRNT Rising Edge (Figure 10)	 t_{TKD1} t_{TKD2}	 – –	 400 190	ns
IRQ/DEND Delay Time ϕ2 DMA Falling Edge (Figure 8) DGRNT Rising Edge (Figure 10)	 t_{DED1} t_{DED2}	 – –	 300 190	ns

FIGURE 3 – EXPANDED BLOCK DIAGRAM

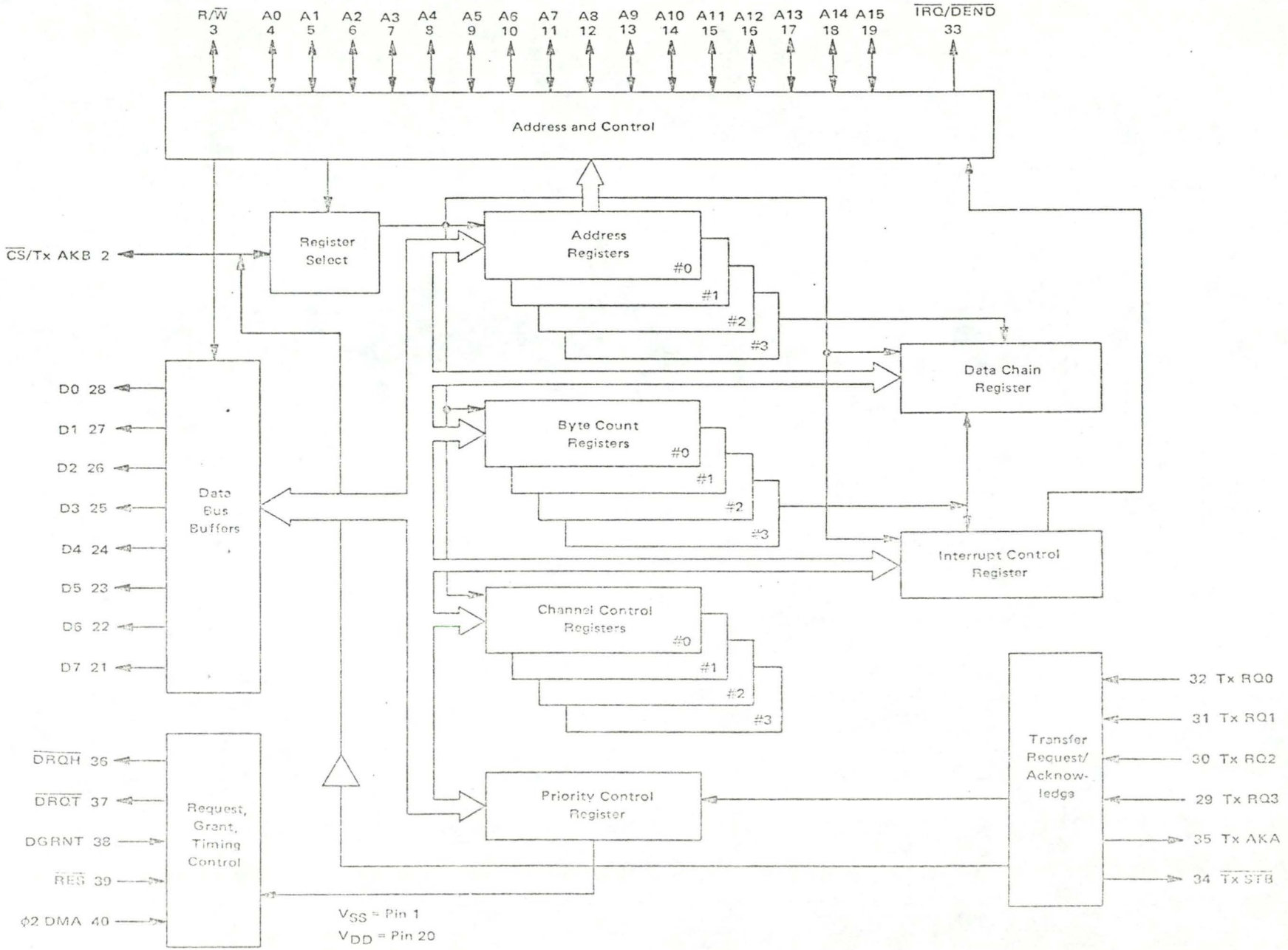


FIGURE 4 – READ/WRITE OPERATION SEQUENCE

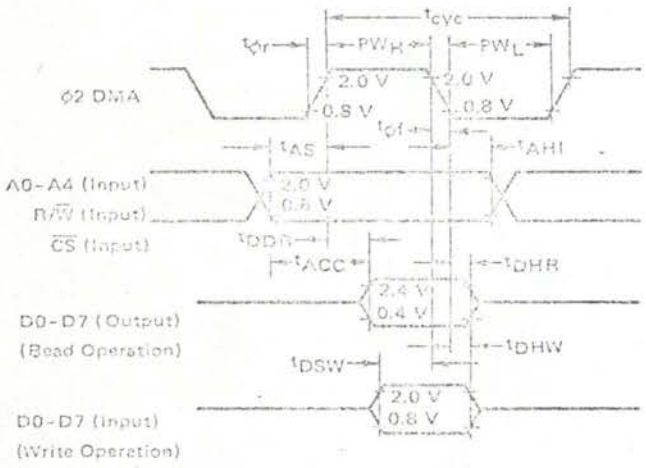


FIGURE 5 – Tx RQ INPUT TIMING

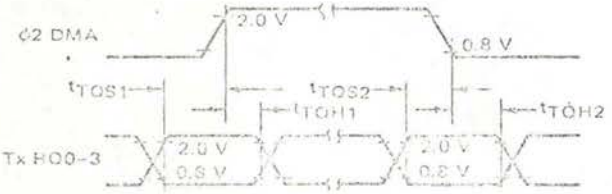


FIGURE 6 – DGRNT INPUT TIMING

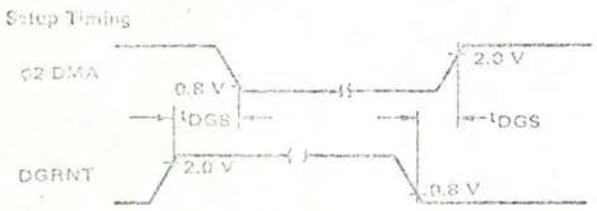


FIGURE 7 – DRQH, DRQT, Tx AK OUTPUT TIMING

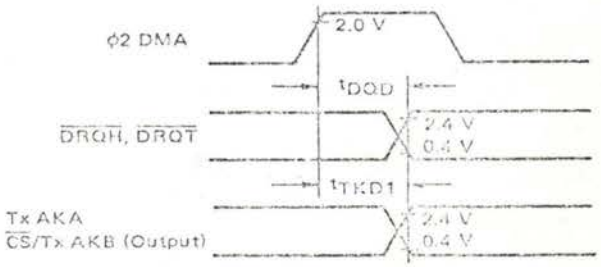


FIGURE 8 – ADDRESS, IRQ/DEND OUTPUT TIMING

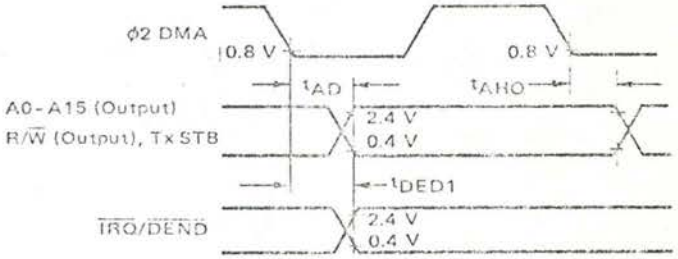


FIGURE 9 – ADDRESS THREE-STATE TIMING

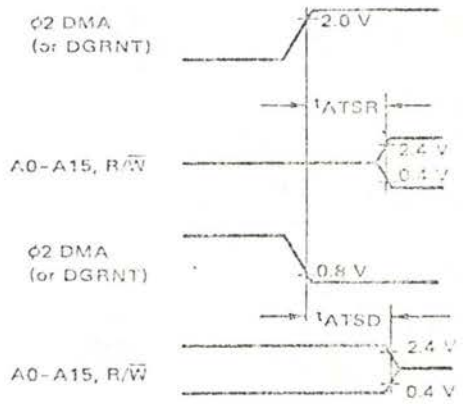


FIGURE 10 – Tx AKB, IRQ/DEND OUTPUT TIMING FROM DGRNT INPUT

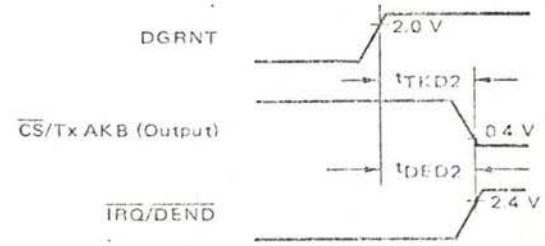
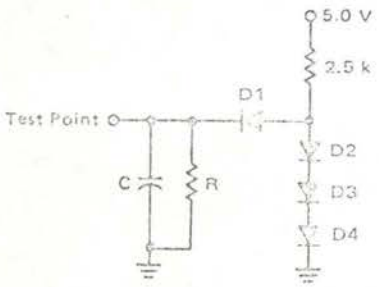
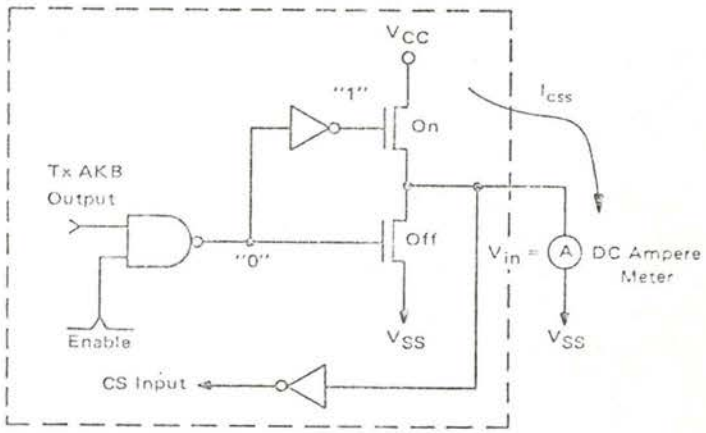


FIGURE 11 – TEST LOADS



Test Pin	C = pF	R = kΩ
D0-D7	130	11.7
A0-A15, R/W	90	16.5
CS/Tx AKB	50	24
Others	30	24

FIGURE 12 – CS/Tx AKB SOURCE CURRENT TEST CIRCUIT



DEVICE OPERATION

The DMAC has fifteen addressable registers, eight of these are sixteen bits in length. Each channel has a separate Address Register and a Byte Count Register, each of which is sixteen bits. There are also four Channel Control Registers. The three General Control Registers common to all four channels are the Priority Control Register, the Interrupt Control Register, and the Data Chain Register.

To prepare a channel for DMA, the Address Registers must be loaded with the starting memory address and the Byte Count Register loaded with the number of bytes to be transferred. The bits in the Channel Control Register establish the direction of the transfer, the mode, and the address increment or decrement after each cycle. Each channel can be set for one of three transfer modes: Three-State Control (TSC) Steal, Halt Steal, or Halt Burst. Two read-only status bits in the Channel Control Register indicate when the channel is busy transferring data and when the DMA transfer is completed.

The Priority Control Register enables the transfer requests from the peripheral controllers and establishes either a fixed priority or rotating priority scheme of servicing these requests.

When the DMA transfer for a channel is complete (the Byte Count Register is zero), a DMA End signal is directed to the peripheral controller and an $\overline{IRQ}$ goes to the MPU. Enabling of these interrupts is done in the Interrupt Control Register. The DMA End/$\overline{IRQ}$ flag bit is read from this register.

Chaining of data transfers is controlled by the Data Chain Register. When enabled, the contents of the Address and Byte Count Registers for channel #3 are put into the registers of the channel selected for chaining when its Byte Count Register becomes zero. This allows for repetitively reading or writing a block of memory.

During the DMA mode, the DMAC controls the address bus and data bus for the system as well as providing the R/$\overline{W}$ line and a signal to be used as VMA. When a peripheral device controller desires a DMA transfer, it is requested by a Transfer Request. Assuming this request is enabled and meets the test of highest priority, the DMAC will issue a DMA Request. When the DMAC receives the DMA Grant, it gives a Transfer Acknowledge to the peripheral device controller, at which time the data is transferred. When the channel's Byte Count Register equals zero, the transfer is complete and a DMA End is given to the peripheral device controller, and an $\overline{IRQ}$ is given to the MPU.

Initialization

During a power-on sequence, the DMAC is reset via the $\overline{RES}$ input. All registers, with the exception of the Address and Byte Count Registers, are set to a logic "0" state. This disables all requests and the Data Chain function while masking all interrupts. The Address, Byte Count, and Channel Control Registers must be programmed before the respective transfer request bit is enabled in the Priority Control Register.

Transfer Modes

There are three ways in which a DMA transfer may be done. The one used is determined by the data transfer rate required, the number of channels attached, and the hardware complexity allowable. Refer to Figures 13 through 15.

Two of the modes, TSC Steal and Halt Steal, are done by cycle- stealing from the MPU. The Three-State Control (TSC) Steal mode is initiated by the DMAC bringing the $\overline{DRQT}$ line LOW. This line goes to the system clock driver which returns a HIGH on DGRNT on the rising edge of the system $\phi 1$ clock. The DGRNT signal must cause the address control and data lines to go to the high impedance state. The DMAC now supplies the address from the Address Register of the channel requesting. It also supplies the R/$\overline{W}$ signal as determined from the Channel Control Register. After one byte is transferred, control is restored to the MPU. This method stretches the $\phi 1$ and $\phi 2$ clocks while the DMAC uses the memory.

The second method of cycle-stealing is the Halt Steal mode. This method actually halts the MPU instead of stretching the $\phi 1$ clock for the transfer period. This mode is initiated by the DMAC bringing the $\overline{DRQR}$ line LOW. This line connects to the MPU $\overline{HALT}$ input. The MPU Bus Available (BA) line is the DGRNT input to the DMAC. While the MPU is halted, its Address Bus, Data Bus, and R/$\overline{W}$ are in the high impedance state. The DMAC now supplies the address and R/$\overline{W}$ line. After one byte is transferred, the $\overline{HALT}$ line is returned HIGH and the MPU regains control. In this mode, the MPU stops internal activity and is removed from the system while the DMAC uses the memory.

The third mode of transfer is the Halt Burst mode. This mode is similar to the Halt Steal mode, except that the transfer does not stop with one byte. The MPU is halted while an entire block of data is transferred. When the channel's Byte Count Register equals zero, the transfer is complete and control is returned to the MPU. This mode gives the highest data transfer rate, at the expense of the MPU being inactive during the transfer period.

FIGURE 13 – TSC STEAL MODE

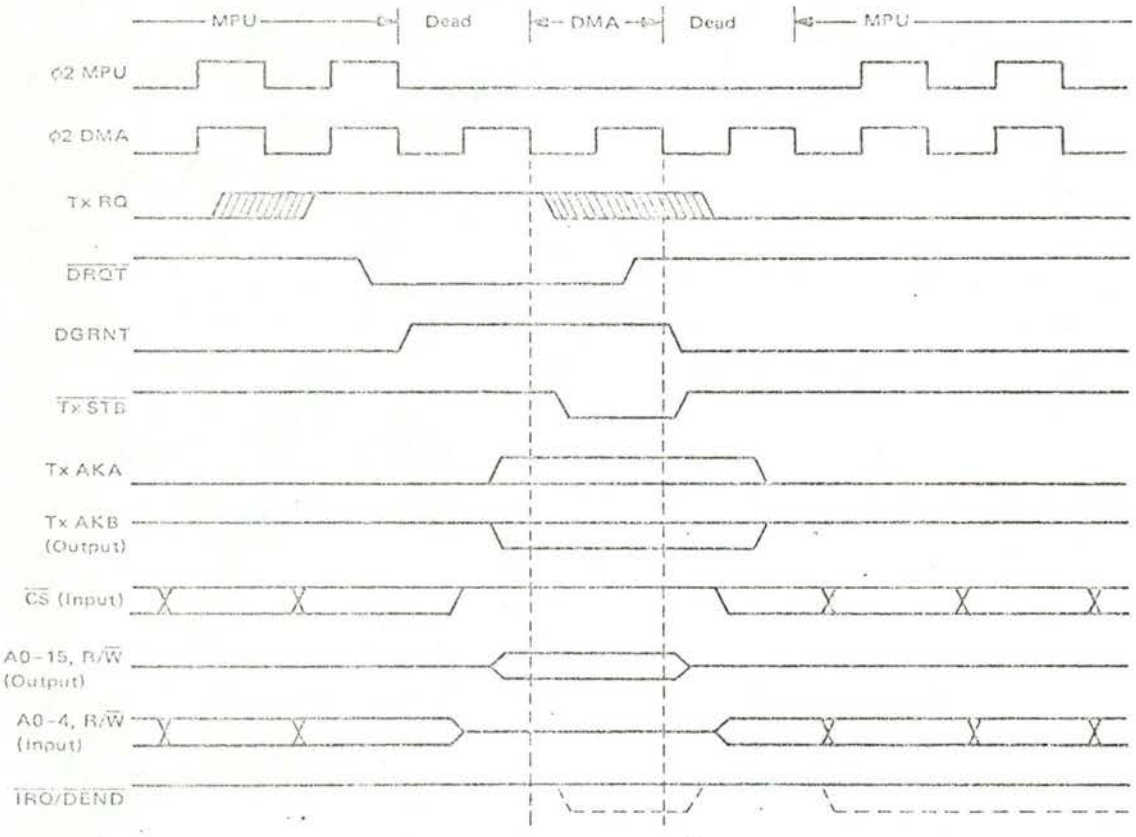


FIGURE 14 – HALT STEAL MODE

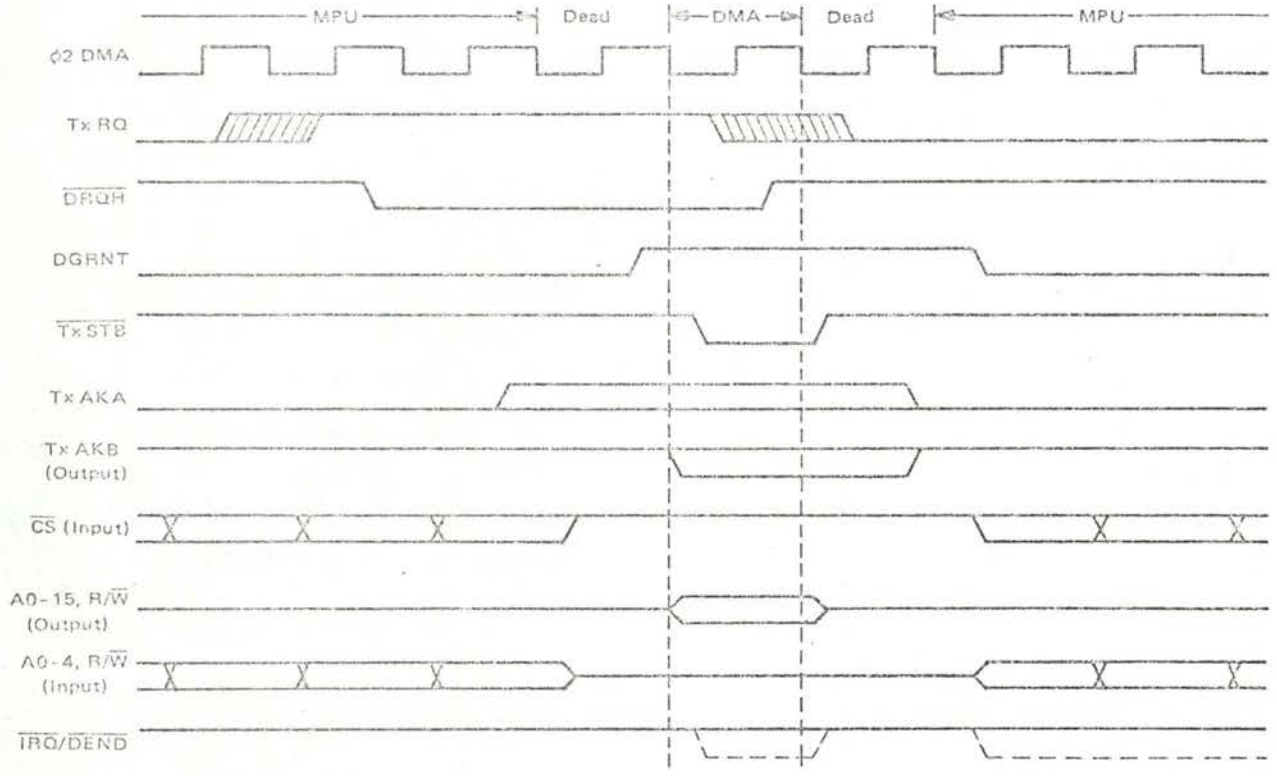
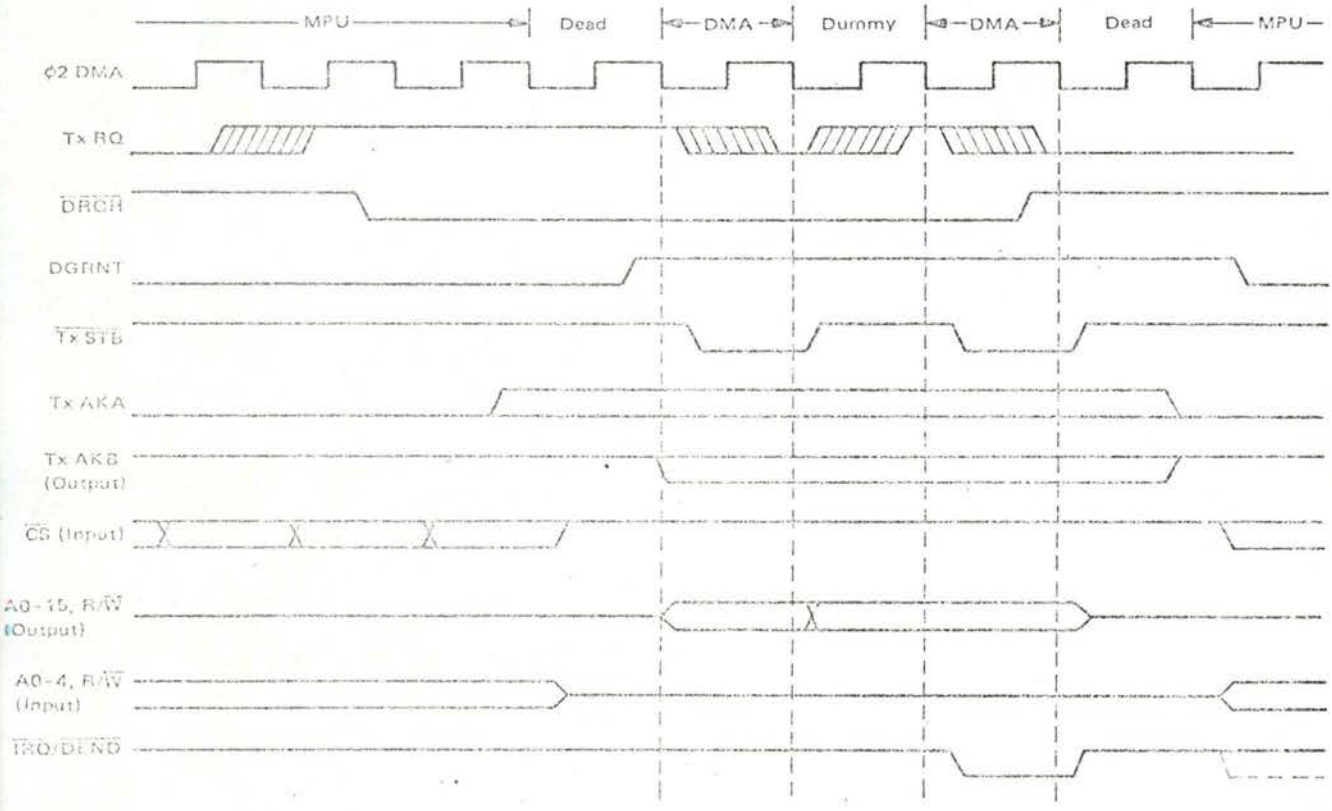


FIGURE 15 – HALT BURST MODE



DMAC Interface Signals for the MPU

The DMAC interfaces with the M6800 MPU through the eight-bit bidirectional data bus, the Chip Select line, five address lines, an Interrupt Request line, the Read/Write line, and the Reset line. These signals, in conjunction with the M6800 VMA output, permit the MPU to have access to the DMAC. Four other lines associated with the MPU and the clock driver are the DRQT, DRQH, DGRNT, and the ϕ2 DMA.

Bidirectional Data (D0-D7). The Bidirectional Data lines (D0-D7) allow for data transfer between the DMAC and the MPU. The data bus output drivers are three-state devices that remain in the high impedance state except when the MPU performs DMAC read operations.

Chip Select/Transfer Acknowledge B ($\overline{CS}$/Tx AKB). This line is multiplexed, serving both as an input and an output. $\overline{CS}$/Tx AKB is an output in the four-channel mode during the DMA transfer. At all other times, it is a high impedance TTL compatible input used to address the DMAC. The DMAC is selected when $\overline{CS}$/Tx AKC is LOW. VMA must be used in generating this input to insure that false selects will not occur. Transfers of data to and from the DMAC are then performed under the control of the ϕ2 DMA, Read/Write, and A0-A4 address lines. In the four-channel mode when Tx AKB is needed, the $\overline{CS}$ gate must have an open-collector output (a pull-up resistor should not be used). In the two-channel mode, $\overline{CS}$/Tx AKB is always an input.

Address Lines A0-A4 (A0-A4). Address lines A0-A4 are both input and output lines. In the MPU mode, these are high impedance inputs used to address the DMAC registers. In the DMA mode, these lines are outputs which are set to the contents of the Address Register of the channel being processed.

Interrupt Request/DMA End ($\overline{IRQ}$/$\overline{DEND}$). Interrupt Request/DMA End is a TTL compatible, active LOW output that is used to interrupt the MPU and to signal the peripheral controller that the data block transfer has ended. If the Interrupt has been enabled, the $\overline{IRQ}$/$\overline{DEND}$ line will go LOW after the last DMA cycle of a transfer. An open collector gate must be connected to DGRNT and $\overline{IRQ}$/$\overline{DEND}$ to prevent false interrupts from the DEND signal when interrupts are not enabled. See Figure 14.

Read/Write (R/$\overline{W}$). Read/Write is a TTL compatible line that is a high impedance input in the MPU mode and an output in the DMA mode. In the MPU mode, it is used to control the direction of data flow through the DMAC's input/output data bus interface. When Read/Write is HIGH (MPU read cycle) and the chip is selected, DMAC data output buffers are turned on and a selected register is read. When it is LOW, the DMAC output drivers are turned off and the MPU writes into a selected register.

In the DMA mode, Read/Write is an output to drive the memory and peripheral controllers. Its state is determined by bit 0 of the Channel Control Register for the channel being serviced. When Read/Write is HIGH, the memory is read and the data from the memory is written into the peripheral controller. When it is LOW, the peripheral controller is read and its data stored in the memory. In the DMA mode, the DMAC data buffers are off, so data is not available on the Data Bus (D0-D7).

Reset ($\overline{RES}$). The $\overline{RES}$ input provides a means of resetting the DMAC from an external source. In the LOW state, the $\overline{RES}$ input causes all registers, with the exception of the Address and Byte Count Registers, to be reset to the logic "0" state. This disables all transfer requests, masks all interrupts, disables the data chain function, and puts each Channel Control Register into the condition of memory write, Halt Steal transfer mode, and address increment.

Transfer Signals to the MPU

Two DMA request output lines and a DMA Grant input line, together with the system clock, synchronize the DMAC with the MPU system.

DMA Request Three-State Control Steal ($\overline{DRQT}$). This active LOW output requests a DMA transfer for a channel configured for the TSC Steal transfer mode. This line is connected to the system clock driver, requesting a ϕ1 clock stretch. It will remain in the LOW state until the transfer has begun.

DMA Request Halt Steal ($\overline{DRQH}$). This active LOW output requests a DMA transfer for a channel programmed for the Halt Steal or Halt Burst mode transfer. This line is connected directly to the MPU $\overline{HALT}$ input and remains LOW until the last byte has begun to be transferred.

DMA Grant (DGRNT). This is a high impedance input to the DMAC, giving it control of the system buses. For the TSC Steal mode, the signal comes from the system clock drive circuit (DMA Grant), indicating that the clock is being stretched. For either of the Halt modes, this signal is the Bus Available from the MPU, indicating that the MPU has halted and turned control of its buses over to the DMAC. For a design involving TSC Steal and Halt mode transfers, this input must be the OR of the clock driven DMA Grant and the MPU BA.

ϕ2 DMA. Transferring in and out of the DMAC registers, sampling of channel request lines and gating of other control signals to the system is done internally in conjunction with the ϕ2 DMA high impedance input. This input must be the system memory clock (non-stretched ϕ2 clock).

Transfer Signals From the Peripheral Controller

Transfer Request (Tx RQ0-3). Each of the four channels has its own high impedance input request for transfer line. The peripheral controller requests a transfer by setting its Tx RQ line HIGH (a logic "1"). The lines are sampled according to the priority and enabling established in the Priority Control Register. In the Steal mode and the first byte of the Halt Burst mode, the Tx RQ signals are tested on the positive edge of $\phi 2$ DMA and the highest priority channel is strobed. Once strobed, the Tx RQs are not tested until that channel's data transfer is finished. In the succeeding bytes of the Halt Burst mode transfer, the Tx RQ is tested on the negative edge of $\phi 2$ DMA, and data is transferred on the next $\phi 2$ DMA cycle if Tx RQ is HIGH.

Transfer Signals to the Peripheral Controller

Two encoded lines select the channel to be serviced. A strobe line acknowledges the request and performs the transfer. The DMA End line signals to the peripheral controller that the DMA transfer is completed.

Transfer Acknowledge A (Tx AKA). The Transfer Acknowledge A (Tx AKA) is a TTL compatible output used in conjunction with the $\overline{CS}$/Tx AKB line to select the channel to be strobed for transfer and to give the DMA End Signal. In the two-channel mode, only Tx AKA is used to select channel #0 or channel #1, and $\overline{CS}$/Tx AKB is always an input.

Chip Select/Transfer Acknowledge B ($\overline{CS}$/Tx AKB). In the DMA mode, this dual purpose line is encoded together with Tx AKA to select the channel being serviced. Table 1 shows the encoding order.

TABLE 1 – ENCODING ORDER

$\overline{CS}$/Tx AKB	Tx AKA	Channel #
0	0	0
0	1	1
1	0	2
1	1	3

Transfer Strobe (Tx $\overline{STB}$). The Transfer Strobe causes acknowledgement to be given to the peripheral controller and transfers the data to or from the memory. This line is also intended to be the VMA signal for the system in the DMA mode. In a one-channel system, Tx $\overline{STB}$ may be inverted and run to the peripheral controller's Acknowledge input. In a two- or four-channel system, Tx $\overline{STB}$ enables the decode of Tx AKA and $\overline{CS}$/Tx AKB to select the device controller to be acknowledged.

Interrupt Request/DMA End ($\overline{IRQ}$/$\overline{DEND}$). In the DMA mode, this dual purpose line is LOW for the last byte of transfer, indicating a DMA End. This occurs when the Byte Count register decrements to zero.

This line, through the decode of Tx AKA and $\overline{CS}$/Tx AKB, can be used to strobe a DMA End to each device controller.

Address Lines to the Memory

Address Lines A0-A15 (A0-A15). These output lines are in the high impedance state during the MPU mode. In the DMA mode, these lines are outputs which are set to the contents of the Address Register of the channel being processed.

THE DMAC REGISTERS

All of the fifteen registers in the DMAC are Read/$\overline{Write}$ registers, although some of the bits are read only status bits.

Address Registers

Each channel has its own individual 16-bit Address Register. Before a DMA transfer is begun, the starting address for the transfer must be loaded into the Address Register. Depending on the state of the Channel Control Register, bit 3, the Address Register will be decremented or incremented after each byte of transfer.

Byte Count Registers

Each channel also has its own Byte Count Register. Before the DMA transfer, this register must be loaded with the number of bytes to be transferred. Since it is 16 bits in length, the transfer can be up to 65,536 bytes of data. The Byte Count Register is decremented at the beginning of each DMA cycle.

Channel Control Registers

The control of each channel's DMA transfer is programmed into its Channel Control Register. Bits 4 and 5 are unused.

Read/$\overline{Write}$ (R/$\overline{W}$), Bit 0. The direction of the DMA transfer is controlled by this bit. When it is HIGH, the peripheral controller reads the memory. When it is LOW, the transfer will be in the opposite direction, thus writing into the memory. The system R/$\overline{W}$ line is in the same state as this R/$\overline{W}$ bit in the DMA mode. The device controller must change the sense of its R/$\overline{W}$ input during the DMA mode.

Burst/Steal, Bit 1. This bit, along with bit 2, selects the mode of the DMA transfer. With bit 1 HIGH, the Burst mode is selected. A LOW selects the Steal mode. Table 2 shows the mode selection.

TSC/Halt, Bit 2. Bit 2 helps select the mode of DMA transfer. When this bit is HIGH, the TSC mode is selected.

When LOW, the Halt mode is selected. Table 2 shows the mode selection. A TSC Burst mode is illegal for M6800-family processors due to restrictions on $\phi 1$ clock stretching for these products.

TABLE 2 – MODE SELECT

Bit 2	Bit 1	DMA Transfer Mode
0	0	Halt Steal
0	1	Halt Burst
1	0	TSC Steal
1	1	(Illegal)

Address Up/Down, Bit 3. Bit 3 controls the change in the Address Register for each DMA cycle. If this bit is LOW, the Address Register will be incremented each time the Byte Count Register decrements. If the bit is HIGH, the Address Register will be decremented.

Busy/Ready Flag, Bit 6. The Busy/Ready Flag is a read only status bit that indicates a DMA transfer is in process on that channel. This bit goes HIGH at the beginning of the transfer and remains so until the $\overline{IRQ}/\overline{DEND}$ has been LOW for one cycle (DMA End). The bit is then reset and the channel can again be configured for a transfer.

DMA End Flag (DEND), Bit 7. The DEND bit indicates a DMA block transfer has ended. This bit is set at the same time the Busy/Ready Flag is reset. The DEND bit is reset by the MPU reading the Channel Control Register. This bit causes an interrupt if enabled in the Interrupt Control Register.

PRIORITY CONTROL REGISTER

The enabling and prioritizing of the transfer requests (Tx RQs) are done in the Priority Control Register. Bits 4 through 6 are unused.

Request Enable (REO-3), Bits 0-3. The four channels are individually enabled by setting the respective RE bit HIGH. A LOW on any of these bits disables recognition of the transfer request for that channel. The bit number equals the channel number (i.e., bit 2 equals channel #2).

Rotate Control, Bit 7. The DMAC priority service routine is selected by this Rotate Control bit. When it is LOW, the fixed mode is selected. Channel #0 has the highest priority, channel #1 the next highest, and so on down. When this bit is HIGH, a rotating routine is used. This routine states that initially it will be the same as in the fixed mode. But once a channel has been serviced, it moves to the lowest priority and those that were below it advance to the next highest priority.

Interrupt Control Register

An interrupt is caused by a channel completing its DMA block transfer. DEND (Channel Control Register, Bit 7) flags this condition for each channel. Bits 4 through 6 are unused.

DEND IRQ Enable (DIE0-3) Bits 0-3. Each channel is separately enabled to cause the interrupt. A HIGH enables an interrupt from the channel; a LOW masks the interrupt. The bit number equals the channel number (i.e., bit 2 equals channel #2).

DEND IRQ Flag, Bit 7. This read only bit indicates an IRQ is requested of the MPU when it is HIGH. If the interrupt is enabled (DIE = "1") when a channel's DEND flag (Channel Control Register, bit 7) goes HIGH, the DEND IRQ Flag bit also goes HIGH. It is reset by the MPU reading the Channel Control Register that caused the interrupt.

Data Chain Register

Repetitive reading or writing of a block of memory can be done in the data chain function. A DMA transfer cannot be active on channel #3 during the data chain. Bits 4 through 7 are unused.

Data Chain Enable (DCE), Bit 0. The data chain function is enabled when this bit is HIGH.

Data Chain Channel Selects A, B (DCA, DCB), Bits 1 and 2. The channel updated by data chaining is selected by bits 1 and 2, according to the order shown in Table 3.

TABLE 3 – CHANNEL SELECT

DCB Bit 2	DCA Bit 1	Channel #
0	0	0
0	1	1
1	0	2
1	1	(Illegal)

The data chain function is performed by transferring the contents of Channel #3 Address and Byte Count Registers into the respective registers of the channel selected by bits 1 and 2. This transfer is done during the cycle of $\phi 2$ DMA following the Byte Count Register having decremented to zero.

Two/Four Channel Select (2/4), Bit 3. The DMAC is configured to handle two or four channels by bit 3. This bit HIGH selects the 4-channel mode. In this mode, the $\overline{CS}$/Tx AKB becomes a chip select in the MPU mode and a Transfer Acknowledge B for the DMA mode.

With bit 3 LOW, the 2-channel mode is selected, and the $\overline{CS}$/Tx AKB line is always a chip select, both for the MPU and the DMA mode.

TABLE 4 – DMAC PROGRAMMING MODEL

Register	Address (Hex)	Register Content							
		Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Channel Control	1x*	DMA End Flag (DEND)	Busy/Ready Flag	Not Used	Not Used	Address Up/Down	TSC/ Halt	Burst/ Steal	Read Write (R/$\overline{W}$)
Priority Control	14	Rotate Control	Not Used	Not Used	Not Used	Request Enable #3 (RE3)	Request Enable #2 (RE2)	Request Enable #1 (RE1)	Request Enable #0 (RE0)
Interrupt Control	15	DEND IRQ Flag	Not Used	Not Used	Not Used	DEND IRQ Enable #3 (DIE3)	DEND IRQ Enable #2 (DIE2)	DEND IRQ Enable #1 (DIE1)	DEND IRQ Enable #0 (DIE0)
Data Chain	16	Not Used	Not Used	Not Used	Not Used	Two/Four Channel Select (2/4)	Data Chain Channel Select B	Data Chain Channel Select A	Data Chain Enable

*The x represents the binary equivalent of the channel desired.

Channel Control Register

DEND — Bit 7 – Is set at end of DMA block transfer; reset by MPU reading the Channel Control Register.

Busy/Ready Flag — Bit 6 – Status bit set when in transfer; cleared after DMA End.

Address Up/Down — Bit 3 – HIGH = decrement Address Register for each byte; LOW = increment.

TSC/Halt — Bit 2 – HIGH = select TSC mode; LOW = Halt modes.

Burst/Steal — Bit 1 – HIGH = select Burst mode; LOW = Steal modes.

R/$\overline{W}$ — Bit 0 – HIGH = device controller reads memory; LOW = write into memory.

Priority Control Register

Rotate Control — Bit 7 – HIGH = use rotate routine; LOW = fixed: 0, 1, 2, 3 priority.

RE0-3 — Bits 0-3 – HIGH = enable transfer request for the channel; LOW = request disabled.

Interrupt Control Register

DEND IRQ Flag — Bit 7 – This Flag is set by DENDs in Channel Control Registers when enabled; Reset by reading the Register that caused it to be set.

DIE0-3 — Bits 0-3 – HIGH = enable IRQ by DEND for the channel; LOW = IRQ masked.

Data Chain Register

Two/Four Channel — Bit 3 – HIGH = 4-channel mode; LOW = 2-channel.

Data Chain Channel Select — Bits 2 and 1 – Binary equivalent of channel to be updated by chaining.

Data Chain Enable — Bit 0 – HIGH = enable Data Chain function; LOW = disabled.

Preparation of a channel for a DMA transfer requires:

1. Load the starting address into the Address Register.
2. Load the number of bytes into the Byte Count Register.
3. Program the Channel Control Register for the transfer characteristics: direction (bit 0), mode (bits 1 and 2), and the address update (bit 3).

The channel is now configured. To enable the transfer request, set the appropriate enable bit (bits 0-3) of the Priority Control Register, as well as the Rotate Control bit.

If an interrupt on DMA End is desired, the enable bit (bits 0-3) of the Interrupt Control Register must be set.

If data chaining for the channel is necessary, it is programmed into the Data Chain Register and the appropriate data must be written into the Address and Byte Count Registers for channel #3.

A comparison of the response times and maximum transfer rates is shown in Table 5. The values shown are for a system clock rate of 1 MHz.

TABLE 5 – TRANSFER RATES

Mode	Response Time (μs)	Maximum Transfer Rate (μs/Byte)
Halt Burst	3.5 – 15.5*	1
Halt Steal	3.5 – 15.5*	5 – 15*
TSC Steal	2.5 – 3.5	4

*These values will depend on the cycle in process.

The two 8-bit bytes that form the registers in Table 6 are placed in consecutive memory locations, making it very easy to use the MPU index register in programming them.

TABLE 6 – ADDRESS AND BYTE COUNT REGISTERS

Register	Channel	Address (Hex)
Address High	0	0
Address Low	0	1
Byte Count High	0	2
Byte Count Low	0	3
Address High	1	4
Address Low	1	5
Byte Count High	1	6
Byte Count Low	1	7
Address High	2	8
Address Low	2	9
Byte Count High	2	A
Byte Count Low	2	B
Address High	3	C
Address Low	3	D
Byte Count High	3	E
Byte Count Low	3	F

FIGURE 16 – ONE CHANNEL

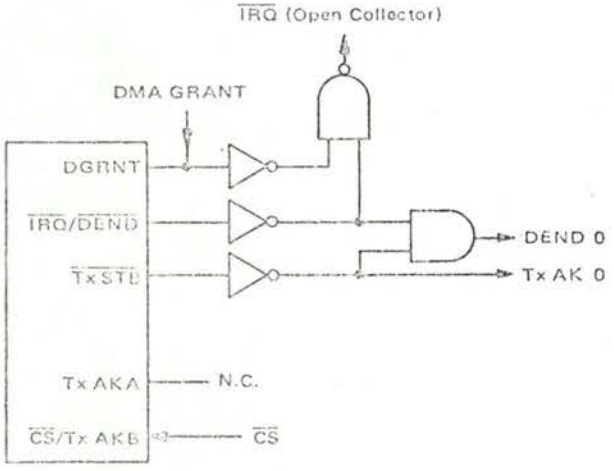


FIGURE 17 – TWO-CHANNEL

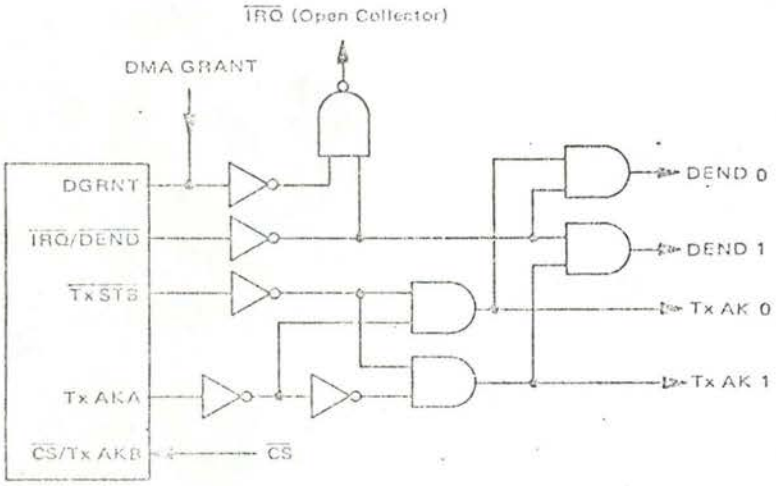
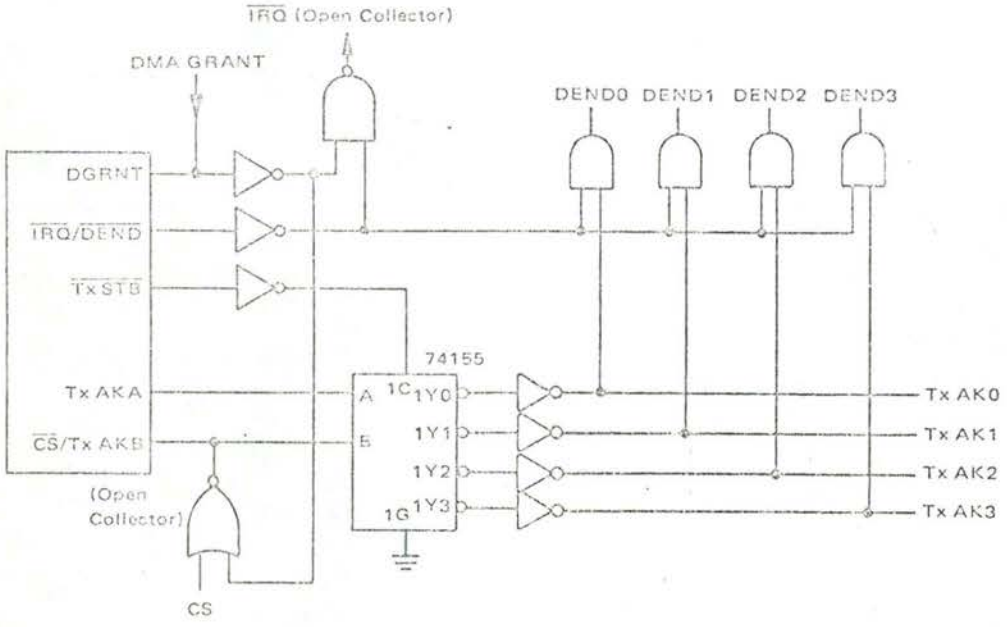


FIGURE 18 – FOUR–CHANNEL



SYSTEM DESCRIPTION

The hardware configuration of the DMAC can be for a 1-, 2-, or 4-channel system. These are shown in Figures 16 through 19.

If the peripheral device controllers do not use the DEND signal, the AND gates generating them are not needed. As mentioned previously, the open collector gate to $\overline{IRQ}$ is to prevent false interrupts from the DEND signal when interrupts are disabled. In the 4-channel mode, the $\overline{CS}$ gate must be open collector so that $\overline{CS}$/Tx AKB can become an output during the DMA cycle.

A typical system design using the MC6800 is shown in Figure 19. System signals are shown on the top; the DMAC control signals are shown on the right. The MC6875 is shown as the clock Generator/Driver. Since the dynamic memory refresh and the DMA control are not separated in the MC6875, it is necessary to have the external Priority Logic to give highest priority to the Refresh Request. Refresh and DMA Grant must not occur during the same cycle.

When using a Halt mode in the DMAC, the MC6875 has no control over the DMA Grant. To prevent a Read or Write during a Refresh cycle, $\phi 2$ DMA must be gated with Refresh Grant.

To be able to use either the TSC Steal and the Halt modes, DGRNT must be the ORed output of Bus Available (BA) and DMA Grant from the Clock Generator/Driver. If only one type is desired, only the one proper line is needed. The MC6875 handles stretching the $\phi 1$ and $\phi 2$ clocks. To three-state the Address and Data Bus and the R/$\overline{W}$ line, the DMA Grant from the Clock Generator/Driver must go to the MC6800's TSC input.

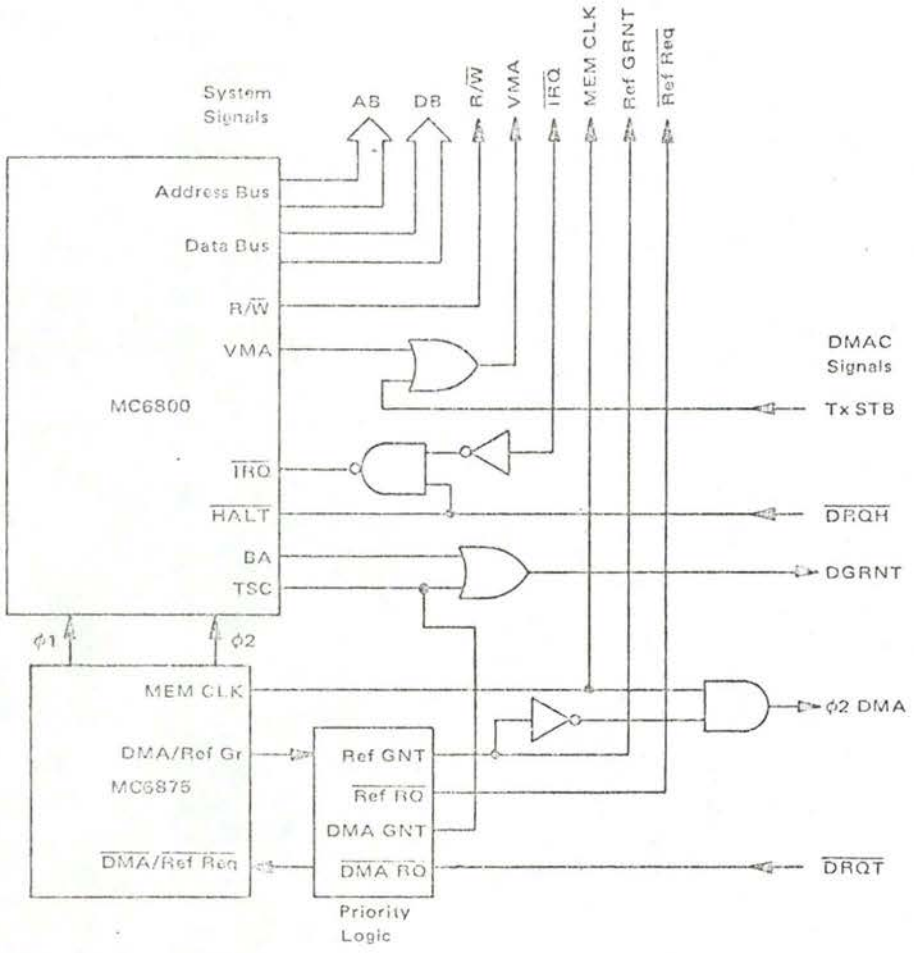
The DMAC's $\overline{DRQH}$ line connects directly to the $\overline{HALT}$ input.

Figure 19 also shows the system $\overline{IRQ}$ is gated with the $\overline{HALT}$ input to get the MPU $\overline{IRQ}$ input. This is necessary only if the Halt modes of transfer are used and the WAI instruction is used along with other system interrupts. If any one of these three is not valid, the system $\overline{IRQ}$ may be connected directly to the MPU $\overline{IRQ}$.

During the DMA cycle, a system VMA signal must be generated by the DMAC. This is done by ORing Tx STB and the MPU VMA line. Another method would be to three-state the MPU VMA by the TSC line input and three-state the Tx STB line by the DGRNT line.

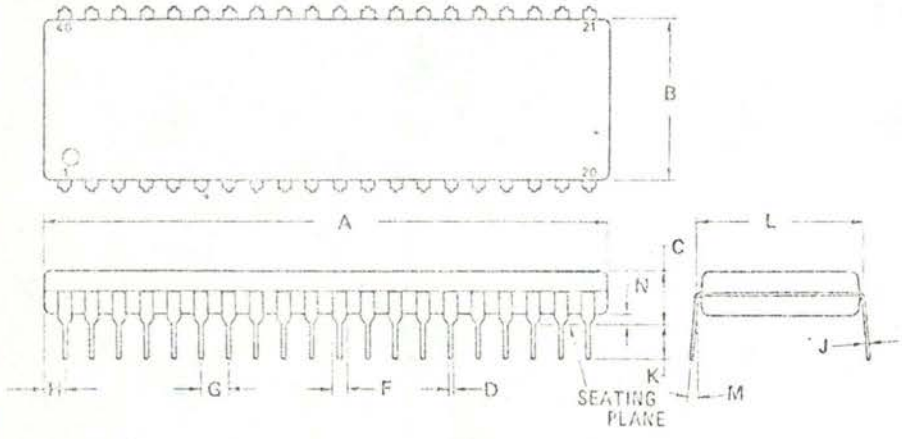
The above gate and line explanations should make it possible to put the DMAC into any system.

FIGURE 19 – TYPICAL SYSTEM



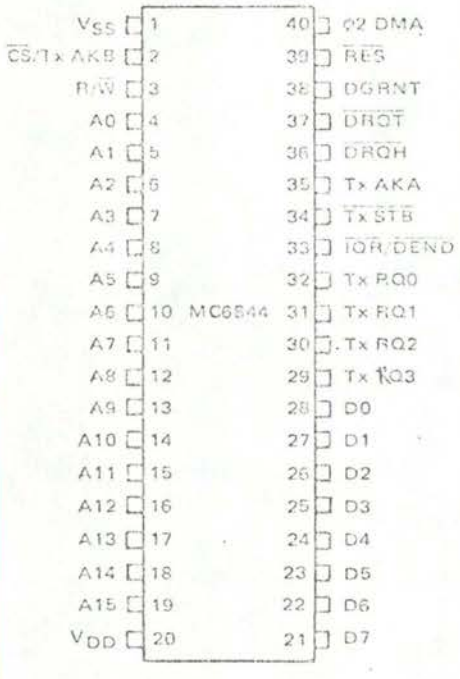
PACKAGE DIMENSIONS

CASE 711-01



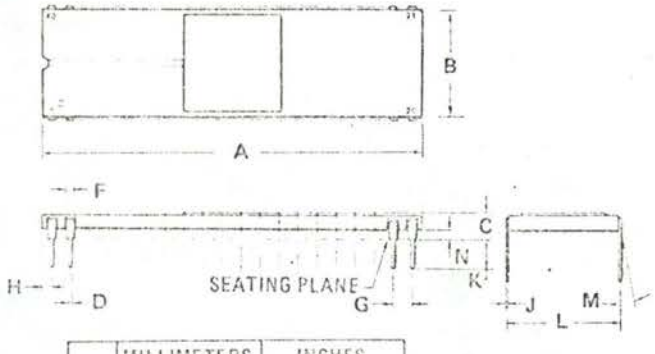
DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	51.82	52.32	2.040	2.060
B	13.72	14.22	0.540	0.560
C	4.57	5.08	0.180	0.200
D	0.35	0.51	0.014	0.020
F	1.07	1.52	0.042	0.060
G	2.41	2.67	0.095	0.105
H	1.65	2.16	0.065	0.085
J	0.20	0.30	0.008	0.012
K	3.68	4.19	0.145	0.165
L	14.99	15.49	0.590	0.610
M	0°	10°	0°	10°
N	0.51	1.02	0.020	0.040

PIN ASSIGNMENT



PACKAGE DIMENSIONS

CASE 715-02
(CERAMIC)



DIM	MILLIMETERS MIN	MILLIMETERS MAX	INCHES MIN	INCHES MAX
A	50.29	51.31	1.980	2.020
B	14.86	15.62	0.585	0.615
C	2.54	4.19	0.100	0.165
D	0.36	0.53	0.015	0.021
F	0.76	1.40	0.030	0.055
G	2.54 BSC		0.100 BSC	
H	0.76	1.78	0.030	0.070
J	0.20	0.33	0.008	0.013
K	2.54	4.19	0.100	0.165
L	14.60	15.37	0.575	0.605
M	–	10°	–	10°
N	0.51	1.52	0.020	0.060

NOTE:
1. LEADS, TRUE POSITIONED WITHIN 0.25 mm (0.010) DIA (AT SEATING PLANE), AT MAX. MAT'L CONDITION.

ANNEXE C :

Version de test du programme

du sous-niveau trame 1

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13     *         Registres PIA
   14     *
   15     ORAH      EQU       X'4000
   16     DDRAH     EQU       X'4000
   17     ORAB      EQU       X'4001
   18     DDRAB     EQU       X'4001
   19     CRAH      EQU       X'4002
   20     CRAB      EQU       X'4003
   21     DDRBH     EQU       X'4004
   22     ORBH      EQU       X'4004
   23     DDRBB     EQU       X'4005
   24     ORBB      EQU       X'4005
   25     CRBH      EQU       X'4006
   26     CRBB      EQU       X'4007
   27     *
   28     *         Registres DMA
   29     *
   30     AR0H      EQU       X'DFE0
   31     AR0L      EQU       X'DFE1
   32     BCR0H     EQU       X'DFE2
   33     BCR0L     EQU       X'DFE3
   34     AR1H      EQU       X'DFE4
   35     AR1L      EQU       X'DFE5
   36     BCR1H     EQU       X'DFE6
   37     BCR1L     EQU       X'DFE7
   38     AR2H      EQU       X'DFE8
   39     AR2L      EQU       X'DFE9
   40     BCR2H     EQU       X'DFEA
   41     BCR2L     EQU       X'DFEB
   42     AR3H      EQU       X'DFEC
   43     AR3L      EQU       X'DFED
   44     BCR3H     EQU       X'DFEE
   45     BCR3L     EQU       X'DFEF
   46     CCR0      EQU       X'DFF0
   47     CCR1      EQU       X'DFF1
   48     CCR2      EQU       X'DFF2
   49     CCR3      EQU       X'DFF3
   50     PC        EQU       X'DFF4
   51     IC        EQU       X'DFF5
   52     DCR       EQU       X'DFF6
   53     *
   54     *         Autres zones
   55     *
   56     FLAG      EQU       0
   57     RR        EQU       1
   58     RNR       EQU       2
   59     REJ       EQU       3
   60     SARM      EQU       5
```

```
 61     DISC     EQU     6
 62     UA       EQU     7
 63     CMDR     EQU     8
 64     I        EQU     9
 65     I_BIS    EQU     10
 66     K        EQU     7
 67     F        EQU     16
 68     ZERROR   ZMB     1
 69     TSTESS   ZMB     1
 70     TRB      ZMB     1
 71     ZTRAV    ZMB     2
 72     ZONTRA   ZMB     1
 73     AD_LOC   EQU     3
 74     AD_REM   EQU     1
 75     BOSIZE   EQU     1536
 76     trbu0    ZMB     140
 77     trbu1    ZMB     140
 78     xmi_sv   ZMB     1
 79     v_send   ZMB     1
 80     intNR    ZMB     1
 81     lastNR   ZMB     1
 82     xmi_bu   ZMB     140
 83     xmi_rq   ZMB     1
 84     v_rec    ZMB     1
 85     xmi_cd   ZMB     1
 86     v_nack   ZMB     1
 87     bo_pct   ZMB     2
 88     bo_opt   ZMB     2
 89     bu_out   ZMB     1536
 90     bo_fct   ZMB     2
 91     window   ZMB     16
 92     bx_ipt   ZMB     2
 93     bo_tpt   ZMB     2
 94     timout   ZMB     1
 95     *
 96     *
 97     *
 98     *
 99     *
100             ORG     X'1000
101     *
102     *       Initialisation SDLC
103     *
104             LDAA    %X'01
105             COMA
106             STAA    CR1
107     *
108     REP107  LDAA    SR
109             COMA
110             BITA    %X'20
111             BEQ     REP107
112     *
113             LDAA    IR
114     *
115             LDAA    %X'01
116             COMA
117             STAA    CR2
118     *
119             LDAA    %X'E1
120             COMA
```

```
121                 STAA    CR1
122     *
123     REP106      LDAA    IR
124                 COMA
125                 BITA    %X'02
126                 BEQ     REP106
127     *
128     *           Initialisation DMA
129     *
130                 LDAA    %X'08
131                 STAA    DCR
132     *
133                 LDAA    %X'04
134                 STAA    CCR1
135     *
136                 INCA
137                 STAA    CCR0
138     *
139                 LDX     %trbu0
140                 INX
141                 STX     AR1H
142     *
143                 LDAA    %145
144                 STAA    BCR1L
145                 CLR     BCR1H
146     *
147                 CLR     TRB
148     *
149                 LDAA    PC
150                 ORAA    %X'02
151                 STAA    PC
152     *
153                 LDAA    IC
154                 ORAA    %X'03
155                 STAA    IC
156     *
157     *
158     ********
159     ********
160                 CLR     ZERROR
161                 LDS     %X'D3FF
162                 LDX     %INTERU
163                 STX     X'DF00
164     *
165     *
166                 CLI
167     WAIT        WAI
168                 BRA     WAIT
169     *
170     INTERU      LDAA    IR
171                 COMA
172                 BITA    %1
173                 BEQ     TSTDMA
174     INTRQ1      PSHA
175                 BITA    %X'80
176                 BEQ     SUITE1
177                 JSR     RENDER
178                 BRA     SUITE2
179     SUITE1      BITA    %X'40
180                 BEQ     SUITE2
```

```
 191              JSR     REWIER
 182    SUITE2    PULA
 183              BITA    %X'20
 184              BEQ     SUITE3
 185              JSR     XMNOER
 186              BRA     TSTDMA
 187    SUITE3    BITA    %X'10
 188              BEQ     TSTDMA
 189              JSR     XMWIUN
 190    TSTDMA    LDAA    IC
 191              BITA    %X'80
R192              BEQ     RETOUR
 193    TSTCR1    LDAA    CCR1
 194              BITA    %X'80
 195              BEQ     TSTCRO
 196              JSR     CASCR1
 197    TSCCRO    LDAA    CCRO
 198              BITA    %X'80
 199              BEQ     RETOUR
 200              JSR     CASCRO
 201              BRA     RETOUR
 202    TSTCRO    LDAA    CCRO
 203              BITA    %X'80
 204              BEQ     ERROR1
 205              JSR     CASCRO
 206    RETOUR    RTI
 207    *
 208    *
 209    *
 210    ERROR1    LDAA    %X'01
 211              STAA    ZERROR
 212              JMP     X'F800
 213    *
 214    *
 215    *  fin de reception sans erreur
 216    *
 217    *
 218    RENOER    LDAA    PC
 219              ANDA    %X'FD
 220              STAA    PC
 221    *
 222              LDAA    %145
 223              SUBA    BCR1L
 224              SUBA    %2
 225    *
 226              LDAB    TRB
 227              BNE     BUF1
 228    *
 229              STAA    trbu0
 230              INC     TRB
 231              LDX     %trbu1
 232    *
 233              LDAA    ()
 234              BNE     ERROR2
 235              BRA     DMAREC
 236    *
 237    BUF1      STAA    trbu1
 238              DEC     TRB
 239              LDX     %trbu0
 240    *
```

```
241             LDAA    ()
242             BNE     ERROR2
243     *
244     DMAREC  INX
245             STX     AR1H
246             LDAA    %145
247             STAA    BCR1L
248             CLR     BCR1H
249             LDAA    PC
250             ORAA    %X'02
251             STAA    PC
252             RTS
253     *
254     *
255     *
256     ERROR2  LDAA    %X'02
257             STAA    ZERROR
258             JMP     X'F800
259     *
260     *
261     *  fin de reception avec erreur
262     *
263     *
264     REWIER  LDAA    PC
265             SWI
266             ANDA    %X'FD
267             STAA    PC
268     *
269             LDAA    TRB
270             BNE     BUFFE1
271             LDX     %trbu0
272             JSR     DMAREC
273             RTS
274     BUFFE1  LDX     %trbu1
275             JSR     DMAREC
276             RTS
277     *
278     *
279     *  fin d'emission sans erreur
280     *
281     *
282     XMNOER  LDAA    xmi_sv
283             ANDA    %X'EF
284             CMPA    %FLAG
285             BEQ     CASFLG
286     *
287     *
288     *
289     CASFCS  CMPA    %SARM
290             BEQ     CASFLG
291             CMPA    %DISC
292             BEQ     CASFLG
293             CMPA    %UA
294             BEQ     CASFLG
295             CMPA    %CMDR
296             BEQ     CASFLG
297     *
298             CMPA    %RR
299             BEQ     MAJNR
300             CMPA    %RNR
```

```
301                 BEQ     MAJNR
302                 CMPA    %REJ
303                 BEQ     MAJNR
304     *
305                 CMPA    %I_BIS
306                 BEQ     TSTTIM
307     *
308                 CMPA    %I
309                 BEQ     INVSEN
310     *
311     *
312     *
313                 LDAA    %X'04
314                 STAA    ZERROR
315                 JMP     X'F800
316     *
317     *
318     *
319     INVSEN      LDAA    v_send
320                 INCA
321                 ANDA    %X'07
322                 STAA    v_send
323     *
324     TSTTIM      LDAA    CRAH
325                 ANDA    %X'08
326                 BEQ     MAJNR
327                 JSR     startT
328     *
329     MAJNR       LDAA    intNR
330                 STAA    lastNR
331     *
332     *
333     *
334     CASFLG      LDX     %xmi_bu
335     *
336                 LDAA    xmi_rq
337                 BNE     SUIVAN
338                 JMP     ESSAI
339     *
340     SUIVAN      STAA    xmi_sv
341                 CLR     xmi_rq
342     *
343                 LDAB    xmi_sv
344                 ANDB    %X'EF
345                 CMPB    %RR
346                 BNE     T1
347                 JMP     TRARR
348     T1          CMPB    %RNR
349                 BNE     T2
350                 JMP     TRARNR
351     T2          CMPB    %REJ
352                 BNE     T3
353                 JMP     TRAREJ
354     T3          CMPB    %SARM
355                 BNE     T4
356                 JMP     TRASRM
357     T4          CMPB    %DISC
358                 BNE     T5
359                 JMP     TRADSC
360     T5          CMPB    %UA
```

```
361             BNE     T6
362             JMP     TRAUA
363     T6      CMPB    %CMDR
364             BNE     ERROR5
365             JMP     TRACDR
366     *
367     *
368     *
369     ERROR5  LDAA    %X'05
370             STAA    ZERROR
371             JMP     X'F800
372     *
373     *
374     *
375     TRARR   LDAB    xmi_sv
376     *
377             ANDB    %F
378             STAB    ZONTRA
379     *
380             BNE     ENVRR
381             JSR     ESSTRI
382     *
383             LDAA    TSTESS
384             BNE     DEHORS
385     *
386     ENVRR   JSR     ADRL
387     *
388             LDAA    v_rec
389             ASLA
390             ASLA
391             ASLA
392             ASLA
393             ASLA
394             ORAA    ZONTRA
395             ORAA    %X'01
396     *
397             STAA    ()
398     *
399             LDAA    %X'02
400             STAA    BCROL
401             CLR     BCROH
402             JMP     EMIDMA
403     *
404     DEHORS  RTS
405     *
406     *
407     *
408     TRARNR  JSR     ADRL
409     *
410             LDAA    v_rec
411             ASLA
412             ASLA
413             ASLA
414             ASLA
415             ASLA
416             ORAA    xmi_rq
417             ANDA    %X'F0
418     *
419             ORAA    %X'05
420     *
```

```
421                 STAA    ()
422     *
423                 LDAA    %X'02
424                 STAA    BCROL
425                 CLR     BCROH
426                 JMP     EMIDMA
427     *
428     *
429     *
430     TRAREJ      JSR     ADRL
431     *
432                 LDAA    v_rec
433                 ASLA
434                 ASLA
435                 ASLA
436                 ASLA
437                 ASLA
438                 ORAA    xmi_rq
439                 ANDA    %X'F0
440     *
441                 ORAA    %X'09
442     *
443                 STAA    ()
444     *
445                 LDAA    %X'02
446                 STAA    BCROL
447                 CLR     BCROH
448                 JMP     EMIDMA
449     *
450     *
451     *
452     TRASRM      JSR     ADRR
453     *
454                 LDAA    %X'0F
455                 ORAA    xmi_rq
456     *
457                 STAA    ()
458     *
459                 LDAA    %X'02
460                 STAA    BCROL
461                 CLR     BCROH
462                 JMP     EMIDMA
463     *
464     *
465     *
466     TRADSC      JSR     ADRR
467     *
468                 LDAA    %X'43
469                 ORAA    xmi_rq
470     *
471                 STAA    ()
472     *
473                 LDAA    %X'02
474                 STAA    BCROL
475                 CLR     BCROH
476                 JMP     EMIDMA
477     *
478     *
479     *
480     TRAUA       JSR     ADRL
```

```
481     *
482             LDAA    %X'63
483             ORAA    xmi_rq
484     *
485             STAA    ()
486     *
487             LDAA    %X'02
488             STAA    BCROL
489             CLR     BCROH
490             JMP     EMIDMA
491     *
492     *
493     *
494     TRACDR  JSR     ADRL
495     *
496             LDAA    %X'87
497             ORAA    xmi_rq
498     *
499             STAA    ()
500     *
501             INX
502             CLR     ()
503             INX
504             CLR     ()
505             INX
506             CLR     ()
507     *
508             LDAA    %X'05
509             STAA    BCROL
510             CLR     BCROH
511             JMP     EMIDMA
512     *
513     *
514     *
515     ADRL    LDAA    %AD_LOC
516             STAA    ()
517             INX
518             RTS
519     *
520     *
521     *
522     ADRR    LDAA    %AD_REM
523             STAA    ()
524             INX
525             RTS
526     *
527     *
528     *
529     EMIDMA  LDAA    %X'C1
530             COMA
531             STAA    CR1
532     *
533             LDX     %xmi_bu
534             STX     AROH
535             INC     BCROL
536             LDAA    PC
537             ORAA    %X'01
538             STAA    PC
539             RTS
540     *
```

```
541     *
542     *
543     startT  LDAA    %X'37
544             STAA    CRAH
545             LDAA    %X'01
546             STAA    timout
547             LDAA    %X'3F
548             STAA    CRAH
549             RTS
550     *
551     *
552     *
553     ESSAI   JSR     ESSTRI
554     *
555             LDAA    TSTESS
556     *
557             BNE     SORTIE
558     *
559             LDAA    %FLAG
560             STAA    xmi_sv
561             LDAA    %X'E1
562             COMA
563             STAA    CR1
564     *
565     SORTIE  RTS
566     *
567     *
568     *
569     ESSTRI  LDAA    xmi_cd
570             BNE     C1
571             JMP     CMD0
572     C1      CMPA    %X'01
573             BNE     C2
574             JMP     CMD1
575     C2      CMPA    %X'02
576             BNE     C3
577             JMP     CMD2
578     C3      CMPA    %X'03
579             BNE     ERROR6
580             JMP     CMD3
581     *
582     *
583     *
584     ERROR6  LDAA    %X'06
585             STAA    ZERROR
586             JMP     X'F800
587     *
588     *
589     *
590     CMD0    CLR     TSTESS
591             RTS
592     *
593     *
594     *
595     CMD1    LDAB    v_send
596             SUBB    v_nack
597             ADDB    %X'08
598             ANDB    %X'07
599             LDAA    bo_pct+1
600             CBA
```

```
601             BHI     BON
602             JMP     RATE
603     *
604     BON     LDAA    v_send
605             LDAB    v_nack
606             ADDB    %K
607             ANDB    %X'07
608             CBA
609             BNE     CHARGE
610             JMP     RATE
611     *
612     CHARGE  LDX     bo_opt
613             LDAA    ()
614             BNE     CALFEN
615     *
616             LDAB    %bu_out
617             LDAA    %bu_out/256
618             ADDB    BOSIZE
619             ADCA    BOSIZE/256
620             SUBB    bo_opt+1
621             SBCA    bo_opt
622             ADDB    bo_fct+1
623             ADCA    bo_fct
624             STAB    bo_fct+1
625             STAA    bo_fct
626     *
627             LDX     %bu_out
628             STX     bo_opt
629     *
630     CALFEN  LDAA    v_send
631             ASLA
632             STAA    ZTRAV+1
633             CLR     ZTRAV
634             LDX     ZTRAV
635     *
636             LDAA    bo_opt
637             LDAB    bo_opt+1
638             STAA    window()
639             STAB    window+1()
640     *
641             LDX     %xmi_bu
642             LDAA    %AD_REM
643             STAA    ()
644             INX
645     *
646             LDAA    v_rec
647             ASLA
648             ASLA
649             ASLA
650             ASLA
651             ORAA    v_send
652             ASLA
653             STAA    ()
654             INX
655             STX     bx_ipt
656     *
657             LDX     bo_opt
658             LDAA    ()
659             INX
660             STX     bo_opt
```

```
661     *
662             STAA    BCROL
663             INC     BCROL
664             INC     BCROL
665             CLR     BCROH
666     *
667     BOUCL1  LDX     bo_opt
668             LDAB    ()
669             INX
670             STX     bo_opt
671             LDX     bx_ipt
672             STAB    ()
673             INX
674             STX     bx_ipt
675             DECA
676             BNE     BOUCL1
677     *
678             LDAA    %I
C679            STAA    xmi_sv
680     *
681             CLR     TSTESS
682             INC     TSTESS
683             JSR     EMIDMA
684             RTS
685     *
686     RATE    CLR     TSTESS
687             RTS
688     *
689     *
690     *
691     CMD2    CLR     xmi_cd
692             INC     xmi_cd
693             LDAA    v_nack
694             STAA    v_send
695     *
696             LDAB    v_send
697             ASLB
698     *
699             STAB    ZTRAV+1
700             CLR     ZTRAV
701             LDX     ZTRAV
702             LDAA    window+1()
703             LDAB    window()
704             STAA    bo_opt+1
705             STAB    bo_opt
706     *
707             LDX     %xmi_bu
708             LDAA    %AD_REM
709             STAA    ()
710             INX
711     *
712             LDAA    v_rec
713             ASLA
714             ASLA
715             ASLA
716             ASLA
717             ORAA    v_send
718             ASLA
719             STAA    ()
720             INX
```

```
721             STX     bx_ipt
722     *
723             LDX     bo_opt
724             LDAA    ()
725             INX
726             STX     bo_opt
727     *
728             STAA    BCROL
729             INC     BCROL
730             INC     BCROL
731             CLR     BCROH'
732     *
733     BOUCL2  LDX     bo_opt
734             LDAB    ()
735             INX
736             STX     bo_opt
737             LDX     bx_ipt
738             STAB    ()
739             INX
740             STX     bx_ipt
741             DECA
742             BNE     BOUCL2
743     *
744             LDAA    %I
745             STAA    xmi_sv
746             CLR     TSTESS
747             INC     TSTESS
748             JSR     EMIDMA
749             RTS
750     *
751     *
752     *
753     CMD3    CLR     xmi_cd
754     *
755             LDAB    v_send
756             ASLB
757     *
758             STAB    ZTRAV+1
759             CLR     ZTRAV
760             LDX     ZTRAV
761             LDAA    window+1()
762             LDAB    window()
763             STAA    bo_tpt+1
764             STAB    bo_opt
765     *
766             LDX     %xmi_bu
767             LDAA    %AD_REM
768             STAA    ()
769             INX
770     *
771             LDAA    v_rec
772             ASLA
773             ASLA
774             ASLA
775             ASLA
776             ORAA    v_send
777             ASLA
778             ORAA    %X'10
779             STAA    ()
780             INX
```

```
781                 STX     bx_ipt
782     *
783                 LDX     bo_tpt
784                 LDAA    ()
785                 INX
786                 STX     bo_tpt
787     *
788                 STAA    BCROL
789                 INC     BCROL
790                 INC     BCROL
791                 CLR     BCROH
792     *
793     BOUCL3      LDX     bo_tpt
794                 LDAB    ()
795                 INX
796                 STX     bo_tpt
797                 LDX     bx_ipt
798                 STAB    ()
799                 INX
800                 STX     bx_ipt
801                 DECA
802                 BNE     BOUCL3
803     *
804                 LDAA    %I_BIS
805                 STAA    xmi_sv
806                 CLR     TSTESS
807                 INC     TSTESS
808                 JSR     EMIDMA
809                 RTS
810     *
811     *
812     *  fin d'emission avec erreur
813     *
814     *
815     XMWIUN      LDAA    %X'03
816                 STAA    ZERROR
817                 JMP     X'F800
818     *
819     *
820     *  fin de reception DMA
821     *
822     *
823     CASCR1      LDAA    %X'07
824                 STAA    ZERROR
825                 JMP     X'F800
826     *
827     *
828     *  fin d'emission DMA
829     *
830     *
831     CASCRO      LDAA    CCRO
832                 ANDA    %X'FE
833                 STAA    CCRO
834     *
835                 LDAA    %X'F0
836                 COMA
837                 STAA    CR1
838                 RTS
```

ANNEXE D :

Programmes de tests

du sous-niveau trame 1

initcd.mas

```
 1      *
 2      *       Registres SDLC
 3      *
 4      CR1     EQU     X'5807
 5      CR2     EQU     X'5806
 6      CR3     EQU     X'5805
 7      SR      EQU     X'5802
 8      IR      EQU     X'5803
 9      THR     EQU     X'5803
10      RHR     EQU     X'5804
11      AR      EQU     X'5804
12      *
13      *       Registres PIA ETCD
14      *
15      ORA     EQU     X'5000
16      DDRA    EQU     X'5000
17      CRA     EQU     X'5002
18      *
19              ORG     X'0000
20      *
21              CLR     CRA
22      *
23              LDAA    %X'26
24              STAA    DDRA
25      *
26              LDAB    %X'04
27              STAB    CRA
28      *
29              LDAA    %X'22
30              STAA    ORA
31      *
32              CLR     CRA
33      *
34              LDAA    %X'27
35              STAA    DDRA
36      *
37              STAB    CRA
38      *
39      *
40      *
41      REP105  LDAA    SR
42              COMA
43              BITA    %X'40
44              BEQ     REP105
45      *
46              LDAA    %X'01
47              COMA
48              STAA    CR2
49      *
50              LDAA    %X'C0
51              COMA
52              STAA    CR1
53      *
54              LDX     %X'0040
55      DELAI   DEX
56              NOP
57              BNE     DELAI
58      *
59              LDAA    %X'F0
60              STAA    ORA
```

```
61      *
62                      JMP     X'F800
63                      END
```

Aug 25, 1980 15:19

initd.mas

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13               ORG       X'0000
   14     *
   15               LDAA      %X'01
   16               COMA
   17               STAA      CR1
   18     *
   19     REP107    LDAA      SR
   20               COMA
   21               BITA      %X'20
   22               BEQ       REP107
   23     *
   24               LDAA      IR
   25     *
   26               LDAA      %X'01
   27               COMA
   28               STAA      CR2
   29     *
   30               LDAA      %X'C1
   31               COMA
   32               STAA      CR1
   33     *
   34     REP106    LDAA      IR
   35               COMA
   36               BITA      %X'02
   37               BEQ       REP106
   38     *
   39               JMP       X'F800
   40               END
```

Aug 25, 1980 15:20

reetcd.mas

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13     *         Registres PIA ETCD
   14     *
   15     ORA       EQU       X'5000
   16     DDRA      EQU       X'5000
   17     CRA       EQU       X'5002
   18     *
   19     BUFREC    EQU       X'0200
   20     *
   21               ORG       X'0000
   22     *
   23     *
   24               LDAB      %X'04
   25               STAB      CRA
   26     *
   27               LDAA      %X'22
   28               STAA      ORA
   29     *
   30               CLR       CRA
   31     *
   32               LDAA      %X'27
   33               STAA      DDRA
   34     *
   35               STAB      CRA
   36     *
   37     *
   38     *
   39     REP105    LDAA      SR
   40               COMA
   41               BITA      %X'40
   42               BEQ       REP105
   43     *
   44               LDAA      %X'01
   45               COMA
   46               STAA      CR2
   47     *
   48               LDAA      %X'C0
   49               COMA
   50               STAA      CR1
   51     *
   52               LDX       %X'0040
   53     DELAI     DEX
   54               NOP
   55               BNE       DELAI
   56     *
   57               LDAA      %X'F0
   58               STAA      ORA
   59     *
   60               LDX       %BUFREC
```

```
61      *
62      DRQ11   LDAA    IR
63              COMA
64              BITA    %1
65              BNE     SORTIE
66      *
67              BITA    %4
68              BEQ     DRQ11
69      *
70              LDAA    RHR
71              COMA
72              STAA    ()
73      *
74              INX
75              BRA     DRQ11
76      *
77      SORTIE  JMP     X'F800
78              END
```

```
 1      *
 2      *         Registres SDLC
 3      *
 4      CR1       EQU       X'5807
 5      CR2       EQU       X'5806
 6      CR3       EQU       X'5805
 7      SR        EQU       X'5802
 8      IR        EQU       X'5803
 9      THR       EQU       X'5803
10      RHR       EQU       X'5804
11      AR        EQU       X'5804
12      *
13      BUFEMI    EQU       X'0000
14      *
15                ORG       X'0100
16      LNBUF     FCB       X'55
17      *
18                LDAA      %X'01
19                COMA
20                STAA      CR1
21      *
22      REP107    LDAA      SR
23                COMA
24                BITA      %X'20
25                BEQ       REP107
26      *
27                LDAA      IR
28      *
29                LDAA      %X'01
30                COMA
31                STAA      CR2
32      *
33                LDAA      %X'C1
34                COMA
35                STAA      CR1
36      *
37      REP106    LDAA      IR
38                COMA
39                BITA      %X'02
40                BEQ       REP106
41      *
42                LDAA      %X'33
43                LDX       %BUFEMI
44                LDAB      LNBUF
45      REMPLI    STAA      ()
46                INX
47                DECB
48                BNE       REMPLI
49      *
50                LDX       %BUFEMI
51                LDAB      LNBUF
52      *
53      LOAD      LDAA      ()
54                COMA
55                STAA      THR
56      *
57      DRQ01     LDAA      IR
58                COMA
59                BITA      %2
60                BEQ       DRQ01
```

```
61      *
62              INX
63              DECB
64              BNE     LOAD
65      *
66              LDAA    %X'F0
67              COMA
68              STAA    CR1
69      *
70      INTRQ1  LDAA    IR
71              COMA
72              BITA    %1
73              BEQ     INTRQ1
74      *
75              JMP     X'F800
76              END
```

dmacd3.mas

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13     *         Registres PIA ETCD
   14     *
   15     ORA       EQU       X'5000
   16     DDRA      EQU       X'5000
   17     CRA       EQU       X'5002
   18     *
   19     *         Registres DMA
   20     *
   21     AROH      EQU       X'DFE0
   22     AROL      EQU       X'DFE1
   23     BCROH     EQU       X'DFE2
   24     BCROL     EQU       X'DFE3
   25     AR1H      EQU       X'DFE4
   26     AR1L      EQU       X'DFE5
   27     BCR1H     EQU       X'DFE6
   28     BCR1L     EQU       X'DFE7
   29     AR2H      EQU       X'DFE8
   30     AR2L      EQU       X'DFE9
   31     BCR2H     EQU       X'DFEA
   32     BCR2L     EQU       X'DFEB
   33     AR3H      EQU       X'DFEC
   34     AR3L      EQU       X'DFED
   35     BCR3H     EQU       X'DFEE
   36     BCR3L     EQU       X'DFEF
   37     CCR0      EQU       X'DFF0
   38     CCR1      EQU       X'DFF1
   39     CCR2      EQU       X'DFF2
   40     CCR3      EQU       X'DFF3
   41     PC        EQU       X'DFF4
   42     IC        EQU       X'DFF5
   43     DCR       EQU       X'DFF6
   44     *
   45     *         Autres zones
   46     *
   47     BUFEMI    EQU       X'0200
   48     *
   49     *
   50     *
   51     *
   52     *
   53               ORG       X'0000
   54     *
   55     *         Initialisation PIA ETCD
   56     *
   57               LDAB      %X'04
   58               STAB      CRA
   59     *
   60               LDAA      %X'22
```

```
                STAA    ORA
*
                CLR     CRA
*
                LDAA    %X'27
                STAA    DDRA
*
                STAB    CRA
*
*               Initialisation SDLC
*
REP105          LDAA    SR
                COMA
                BITA    %X'40
                BEQ     REP105
*
                LDAA    %X'01
                COMA
                STAA    CR2
*
                LDAA    %X'C0
                COMA
                STAA    CR1
*
                LDX     %X'0040
DELAI           DEX
                NOP
                BNE     DELAI
*
                LDAA    %X'F0
                STAA    ORA
*
*               Initialisation DMA
*
                LDX     %BUFEMI
                STX     AROH
*
                LDAA    %X'30
                STAA    BCROL
                CLR     BCROH
*
                LDAA    %X'05
                STAA    CCRO
*
                LDAA    %X'08
                STAA    DCR
*
*               Remplissage de BUFEMI
*
                LDX     %BUFEMI
                LDAB    %X'30
                CLRA
REMPLI          STAA    ()
                INX
                INCA
                DECB
                BNE     REMPLI
*
*               Envoi de BUFEMI
*
```

```
121                 LDAA    %X'01
122                 STAA    PC
123                 STAA    IC
124     *
125     ATTIC       LDAA    IC
126                 BITA    %X'80
127                 BEQ     ATTIC
128     *
129     ATTIR       LDAA    IR
130                 COMA
131                 BITA    %2
132                 BEQ     ATTIR
133                 LDAA    %X'F0
134                 COMA
135                 STAA    CR1
136     *
137     INTRQ1      LDAA    IR
138                 COMA
139                 BITA    %1
140                 BEQ     INTRQ1
141     *
142                 JMP     X'F800
143                 END
```

dmatd3.mas

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13     *         Registres DMA
   14     *
   15     AR0H      EQU       X'DFE0
   16     AR0L      EQU       X'DFE1
   17     BCR0H     EQU       X'DFE2
   18     BCR0L     EQU       X'DFE3
   19     AR1H      EQU       X'DFE4
   20     AR1L      EQU       X'DFE5
   21     BCR1H     EQU       X'DFE6
   22     BCR1L     EQU       X'DFE7
   23     AR2H      EQU       X'DFE8
   24     AR2L      EQU       X'DFE9
   25     BCR2H     EQU       X'DFEA
   26     BCR2L     EQU       X'DFEB
   27     AR3H      EQU       X'DFEC
   28     AR3L      EQU       X'DFED
   29     BCR3H     EQU       X'DFEE
   30     BCR3L     EQU       X'DFEF
   31     CCR0      EQU       X'DFF0
   32     CCR1      EQU       X'DFF1
   33     CCR2      EQU       X'DFF2
   34     CCR3      EQU       X'DFF3
   35     PC        EQU       X'DFF4
   36     IC        EQU       X'DFF5
   37     DCR       EQU       X'DFF6
   38     *
   39     *         Autres zones
   40     *
   41     BUFREC    EQU       X'0300
   42     LONG      EQU       X'02FF
   43     *
   44     *
   45     *
   46     *
   47     *
   48               ORG       X'0000
   49     *
   50     *         Initialisation SDLC
   51     *
   52               LDAA      %X'01
   53               COMA
   54               STAA      CR1
   55     *
   56     REP107    LDAA      SR
   57               COMA
   58               BITA      %X'20
   59               BEQ       REP107
   60     *
```

```
 61                 LDAA    IR
 62     *
 63                 LDAA    %X'01
 64                 COMA
 65                 STAA    CR2
 66     *
 67                 LDAA    %X'C1
 68                 COMA
 69                 STAA    CR1
 70     *
 71     REP106      LDAA    IR
 72                 COMA
 73                 BITA    %X'02
 74                 BEQ     REP106
 75     *
 76     *           Initialisation DMA
 77     *
 78                 LDX     %BUFREC
 79                 STX     AR1H
 80     *
 81                 LDAA    %135
 82                 STAA    BCR1L
 83                 CLR     BCR1H
 84     *
 85                 LDAA    %X'04
 86                 STAA    CCR1
 87     *
 88                 LDAA    %X'08
 89                 STAA    DCR
 90     *
 91     *           Reception dans BUFREC
 92     *
 93                 LDAA    %X'02
 94                 STAA    PC
 95                 STAA    IC
 96     *
 97     DRQ11       LDAA    IR
 98                 COMA
 99                 BITA    %1
100                 BEQ     DRQ11
101     *
102                 LDAA    %135
103                 SUBA    BCR1L
104                 STAA    LONG
105     *
106                 JMP     X'F800
107                 END
```

tremcd.mas

```
    1     *
    2     *         Registres SDLC
    3     *
    4     CR1       EQU       X'5807
    5     CR2       EQU       X'5806
    6     CR3       EQU       X'5805
    7     SR        EQU       X'5802
    8     IR        EQU       X'5803
    9     THR       EQU       X'5803
   10     RHR       EQU       X'5804
   11     AR        EQU       X'5804
   12     *
   13     *         Registres PIA
   14     *
   15     ORAH      EQU       X'4000
   16     DDRAH     EQU       X'4000
   17     ORAB      EQU       X'4001
   18     DDRAB     EQU       X'4001
   19     CRAH      EQU       X'4002
   20     CRAB      EQU       X'4003
   21     DDRBH     EQU       X'4004
   22     ORBH      EQU       X'4004
   23     DDRBB     EQU       X'4005
   24     ORBB      EQU       X'4005
   25     CRBH      EQU       X'4006
   26     CRBB      EQU       X'4007
   27     *
   28     *         Registres PIA ETCD
   29     *
   30     ORA       EQU       X'5000
   31     DDRA      EQU       X'5000
   32     CRA       EQU       X'5002
   33     *
   34     *         Registres DMA
   35     *
   36     AR0H      EQU       X'DFE0
   37     AR0L      EQU       X'DFE1
   38     BCR0H     EQU       X'DFE2
   39     BCR0L     EQU       X'DFE3
   40     AR1H      EQU       X'DFE4
   41     AR1L      EQU       X'DFE5
   42     BCR1H     EQU       X'DFE6
   43     BCR1L     EQU       X'DFE7
   44     AR2H      EQU       X'DFE8
   45     AR2L      EQU       X'DFE9
   46     BCR2H     EQU       X'DFEA
   47     BCR2L     EQU       X'DFEB
   48     AR3H      EQU       X'DFEC
   49     AR3L      EQU       X'DFED
   50     BCR3H     EQU       X'DFEE
   51     BCR3L     EQU       X'DFEF
   52     CCR0      EQU       X'DFF0
   53     CCR1      EQU       X'DFF1
   54     CCR2      EQU       X'DFF2
   55     CCR3      EQU       X'DFF3
   56     PC        EQU       X'DFF4
   57     IC        EQU       X'DFF5
   58     DCR       EQU       X'DFF6
   59     *
   60     *         Autres zones
```

```
  61     *
  62     FLAG     EQU     0
  63     RR       EQU     1
  64     RNR      EQU     2
  65     REJ      EQU     3
  66     SARM     EQU     5
  67     DISC     EQU     6
  68     UA       EQU     7
  69     CMDR     EQU     8
  70     I        EQU     9
  71     I_BIS    EQU     10
  72     K        EQU     7
  73     F        EQU     16
  74     ZERROR   ZMB     1
  75     TSTESS   ZMB     1
  76     TRB      ZMB     1
  77     ZTRAV    ZMB     2
  78     ZONTRA   ZMB     1
  79     AD_LOC   EQU     3
  80     AD_REM   EQU     1
  81     BOSIZE   EQU     1536
  82     trbu0    ZMB     140
  83     trbu1    ZMB     140
  84     xmi_sv   ZMB     1
  85     v_send   ZMB     1
  86     intNR    ZMB     1
  87     lastNR   ZMB     1
  88     xmi_bu   ZMB     140
  89     xmi_rq   ZMB     1
  90     v_rec    ZMB     1
  91     xmi_cd   ZMB     1
  92     v_nack   ZMB     1
  93     bo_pct   ZMB     2
  94     bo_opt   ZMB     2
  95     bu_out   ZMB     1536
  96     bo_fct   ZMB     2
  97     window   ZMB     16
  98     bx_ipt   ZMB     2
  99     bo_tpt   ZMB     2
 100     timout   ZMB     1
 101     *
 102     *
 103     *
 104     *
 105     *
 106              ORG     X'1000
 107     *
 108     *        Initialisation PIA ETCD
 109     *
 110              LDAB    %X'04
 111              STAB    CRA
 112     *
 113              LDAA    %X'22
 114              STAA    DRA
 115     *
 116              CLR     CRA
 117     *
 118              LDAA    %X'27
 119              STAA    DDRA
 120     *
```

```
121                 STAB    CRA
122     *
123     *           Initialisation SDLC
124     *
125     REP105      LDAA    SR
126                 COMA
127                 BITA    %X'40
128                 BEQ     REP105
129     *
130                 LDAA    %X'01
131                 COMA
132                 STAA    CR2
133     *
134                 LDAA    %X'C0
135                 COMA
136                 STAA    CR1
137     *
138                 LDX     %X'0040
139     DELAI       DEX
140                 NOP
141                 BNE     DELAI
142     *
143                 LDAA    %X'F0
144                 STAA    ORA
145     *
146     *           Initialisation DMA
147     *
148                 LDAA    %X'08
149                 STAA    DCR
150     *
151                 LDAA    %X'04
152                 STAA    CCR1
153     *
154                 INCA
155                 STAA    CCR0
156     *
157                 LDX     %trbu0
158                 INX
159                 STX     AR1H
160     *
161                 LDAA    %145
162                 STAA    BCR1L
163                 CLR     BCR1H
164     *
165                 CLR     TRB
166     *
167                 LDAA    PC
168                 ORAA    %X'02
169                 STAA    PC
170     *
171                 LDAA    IC
172                 ORAA    %X'03
173                 STAA    IC
174     *
175     *
176     *********
177                 LDAA    %FLAG
178                 STAA    xmi_sv
179                 LDAA    %X'E1
180                 COMA
```

```
181                 STAA    CR1
182                 LDAA    %SARM
183                 STAA    xmi_rq
184                 CLR     xmi_cd
185     ****************
186                 CLR     ZERROR
187                 LDS     %X'D3FF
188                 LDX     %INTERU
189                 STX     X'DF00
190     *
191     *
192                 CLI
193     WAIT        WAI
194                 TST     xmi_rq
195                 BNE     WAIT
196                 LDAA    %UA
197                 STAA    xmi_rq
198                 BRA     WAIT
199     *
200     INTERU      LDAA    IR
201                 COMA
202                 BITA    %1
203                 BEQ     TSTDMA
204     INTRQ1      PSHA
205                 BITA    %X'80
206                 BEQ     SUITE1
207                 JSR     RENOER
208                 BRA     SUITE2
209     SUITE1      BITA    %X'40
210                 BEQ     SUITE2
211                 JSR     REWIER
212     SUITE2      PULA
213                 BITA    %X'20
214                 BEQ     SUITE3
215                 JSR     XMNOER
216                 BRA     TSTDMA
217     SUITE3      BITA    %X'10
218                 BEQ     TSTDMA
219                 JSR     XMWIUN
220     TSTDMA      LDAA    IC
221                 BITA    %X'80
222                 BEQ     RETOUR
223     TSTCR1      LDAA    CCR1
224                 BITA    %X'80
225                 BEQ     TSTCR0
226                 JSR     CASCR1
227     TSCCR0      LDAA    CCR0
228                 BITA    %X'80
229                 BEQ     RETOUR
230                 JSR     CASCR0
231                 BRA     RETOUR
232     TSTCR0      LDAA    CCR0
233                 BITA    %X'80
234                 BEQ     ERROR1
235                 JSR     CASCR0
236     RETOUR      RTI
237     *
238     *
239     *
240     ERROR1      LDAA    %X'01
```

ANNEXE E :

Programmes de l'intégration

- programmes modifiés de SIMTRA

- version de trame 1

```
    1              seg     L
    2     ***********************************************
    3     *                                             *
    4     *   routine de fin de reception sans erreur   *
    5     *                                             *
    6     ***********************************************
    7     *
    8     *   arret du DMA
    9     *
   10     RENDER  LDAA    PC
   11             ANDA    %X'FD'
   12             STAA    PC
   13     *
   14     *   calcul de la longueur de la trame recue
   15     *   (on retire 2 a cause du FCS)
   16     *
   17             LDAA    %145
   18             SUBA    BCR1L
   19             SUBA    %2
   20     *
   21     *   TRB indique dans quel buffer il faut ranger
   22     *   la trame
   23     *
   24             LDAB    TRB
   25             BNE     BUF1
   26     *
   27     *   TRB =0 : on range la longueur dans trbu0
   28     *
r  29             STAA    trbu0
   30             INC     TRB
   31             LDX     %trbu1
   32     *
   33     *   la longueur de l'autre buffer doit etre
   34     *   nulle (remise a 0 par trame2) ; sinon
   35     *   il y a erreur
   36     *
   37             LDAA    ()
   38             BNE     ERROR2
   39             BRA     DMAREC
   40     *
   41     *   TRB =1 : on range la longueur dans trbu1
   42     *
   43     BUF1    STAA    trbu1
   44             DEC     TRB
   45             LDX     %trbu0
   46     *
   47     *   si la longueur de l'autre buffer n'est pas
   48     *   nulle , il y a erreur
   49     *
   50             LDAA    ()
   51             BNE     ERROR2
   52     *
   53     *   on reinitialise l'adresse et le compteur du DMA
   54     *
   55     DMAREC  INX
   56             STX     AR1H
   57             LDAA    %145
   58             STAA    BCR1L
   59             CLR     BCR1H
   60     *
```

```
61      *   on relance le DMA
62      *
63              LDAA    PC
64              ORAA    %X'02
65              STAA    PC
66              RTS
67      *
68      *   cas d'erreur : la longueur du buffer
69      *   n'a pas ete remise a 0 par trame2
70      *
71      ERROR2  LDAA    %X'02
72              STAA    ZERROR
73              JMP     X'F800
74      ************************************************
75      *                                              *
76      *   routine de fin de reception avec erreur    *
77      *                                              *
78      ************************************************
79      *
80      *   arret du DMA
81      *
82      REWIER  LDAA    PC
83              ANDA    %X'FD
84              STAA    PC
85      *
86      *   on relance le DMA en reception sur le meme
87      *   buffer. Cette relance est la meme que dans
88      *   la routine de reception sans erreur en DMAREC.
89      *
90              LDAA    TRB
91              BNE     BUFFE1
92              LDX     %trbu0
93              JSR     DMAREC
94              RTS
95      BUFFE1  LDX     %trbu1
96              JSR     DMAREC
97              RTS
98      ************************************************
99      *                                              *
100     *   routine de fin d ' emission sans erreur    *
101     *                                              *
102     ************************************************
103     *
104     *   on regarde dans xmi_sv pour voir si on avait
105     *   envoye un flag. Si oui, on va tenter d'envoyer
106     *   la trame suivante en CASFLG ; sinon , on
107     *   termine la trame en cours en CASFCS puis on
108     *   tente d'envoyer la trame suivante.
109     *
110     XMNOER  LDAA    xmi_sv
111             ANDA    %X'EF
112             CMPA    %FLAG
113             BEQ     CASFLG
114     *
115     *   on va terminer la trame en cours suivant son
116     *   type.
117     *
118     CASFCS  CMPA    %SARM
119             BEQ     CASFLG
120             CMPA    %DISC
```

```
121             BEQ     CASFLG
122             CMPA    %UA
123             BEQ     CASFLG
124             CMPA    %CMDR
125             BEQ     CASFLG
126     *
127             CMPA    %RR
128             BEQ     MAJNR
129             CMPA    %RNR
130             BEQ     MAJNR
131             CMPA    %REJ .
132             BEQ     MAJNR
133     *
134             CMPA    %I_BIS
135             BEQ     TSTTIM
136     *
137             CMPA    %I
138             BEQ     INVSEN
139     *
140     *   xmi_sv a une valeur anormale : il y
141     *   a erreur
142     *
143             LDAA    %X'04
144             STAA    ZERROR
145             JMP     X'F800
146     *
147     *   incrementation de v_send
148     *
149     INVSEN  LDAA    v_send
150             INCA
151             ANDA    %X'07
152             STAA    v_send
153     *
154     *   enclenchement du timer s'il y a lieu
155     *
156     TSTTIM  LDAA    CRAH
157             ANDA    %X'08
158             BEQ     MAJNR
159             JSR     startT
160     *
161     *   mise a jour de lastNR
162     *
163     MAJNR   LDAA    intNR
164             STAA    lastNR
165     *
166     *   on va tenter d'envoyer la trame suivante.
167     *
168     CASFLG  LDX     %xmi_bu
169     *
170     *   on regarde dans xmi_rq si on peut envoyer
171     *   une trame autre qu'une trame d'information.
172     *   Si c'est non, on essaie d'envoyer une trame d'
173     *   information en allant en ESSAI
174     *
175             LDAA    xmi_rq
176             BNE     SUIVAN
177             JMP     ESSAI
178     *
179     *   si on envoie une trame qui n'est pas d'information,
180     *   on sauve xmi_rq dans xmi_sv et on fait xmi_rq = 0
```

```
181     *
182     SUIVAN  STAA    xmi_sv
183             CLR     xmi_rq
184     *
185     *   suivant la trame a envoyer , on branche a la
186     *   routine adequate pour l'emission.
187     *
188             LDAB    xmi_sv
189             ANDB    %X'EF
190             CMPB    %RR
191             BNE     TO
192             JMP     TRARR
193     TO      CMPB    %RNR
194             BNE     T2
195             JMP     TRARNR
196     T2      CMPB    %REJ
197             BNE     T3
198             JMP     TRAREJ
199     T3      CMPB    %SARM
200             BNE     T4
201             JMP     TRASRM
202     T4      CMPB    %DISC
203             BNE     T5
204             JMP     TRADSC
205     T5      CMPB    %UA
206             BNE     T6
207             JMP     TRAUA
208     T6      CMPB    %CMDR
209             BNE     ERROR5
210             JMP     TRACDR
211     *
212     *   xmi_sv a une valeur anormale : on est dans
213     *   un cas d'erreur.
214     *
215     ERROR5  LDAA    %X'05
216             STAA    ZERROR
217             JMP     X'F800
218     *
219     *   on est dans le cas d'une trame RR . Si le
220     *   bit P = 0 , on va essayer d'envoyer une
221     *   trame d'information . Si on n'y arrive pas
222     *   ou si le bit P = 1 , on envoie un RR .
223     *
224     TRARR   LDAB    xmi_sv
225     *
226             ANDB    %F
227             STAB    ZONTRA
228     *
229             BNE     ENVRR
230             JSR     ESSTRI
231     *   la variable TSTESS nous signale si on a pu envoyer
232     *   une trame d'information . Si oui , on sort . Si
233     *   non , on construit une trame RR .
234     *
235             LDAA    TSTESS
236             BNE     DEHORS
237     *
238     *   on va a la routine de construction de l'adresse .
239     *
240     ENVRR   JSR     ADRL
```

```
241     *
242     *   construction du champs de commande avec v_rec et
243     *   le bit P .
244     *
245             LDAA    v_rec
246             ASLA
247             ASLA
248             ASLA
249             ASLA
250             ASLA
251             ORAA    ZONTRA
252             ORAA    %X'01
253     *
254     *   mise du champs de commande dans le buffer d'emission
255     *
256             STAA    ()
257     *
258     *   preparation du compteur DMA et saut a la routine
259     *   d'emission d'une trame.
260     *
261             LDAA    %X'02
262             STAA    BCROL
263             CLR     BCROH
264             JMP     EMIDMA
265     *
266     *   on a envoye une trame d'information : on sort donc de
267     *   la routine.
268     *
269     DEHORS  RTS
270     *
271     *   on va envoyer une trame RNR.
272     *   on va a la routine de construction de l'adresse.
273     *
274     TRARNR  JSR     ADRL
275     *
276     *   construction du champs de commande avec v_rec et
277     *   le bit P.
278     *
279             LDAA    xmi_sv
280             ANDA    %F
281             LDAA    v_rec
282             ASLA
283             ASLA
284             ASLA
285             ASLA
286             ASLA
287             ORAA    ZONTRA
288             ORAA    %X'05
289     *
290     *   mise du champs de commande dans le buffer d'emission.
291     *
292             STAA    ()
293     *
294     *   preparation du compteur DMA et saut a la routine
295     *   d'emission d'une trame.
296     *
297             LDAA    %X'02
298             STAA    BCROL
299             CLR     BCROH
300             JMP     EMIDMA
```

```
301     *
302     *    on va envoyer une trame REJ.
303     *    on va a la routine de construction de l'adresse.
304     *
305     TRAREJ  JSR     ADRL
306     *
307     *    construction du champs de commande avec v_rec et
308     *    le bit P.
309     *
310              LDAA    xmi_sv
311              ANDA    %F
312              LDAA    v_rec
313              ASLA
314              ASLA
315              ASLA
316              ASLA
317              ASLA
318              ORAA    ZONTRA
319              ORAA    %X'09
320     *
321     *    mise du champs de commande dans le buffer d'emission.
322     *
323              STAA    ()
324     *
325     *    preparation du compteur DMA et saut a la routine
326     *    d'emission d'une trame.
327     *
328              LDAA    %X'02
329              STAA    BCROL
330              CLR     BCROH
331              JMP     EMIDMA
332     *
333     *    on va envoyer une trame SARM.
334     *    on va a la routine de construction de l'adresse.
335     *
336     TRASRM  JSR     ADRR
337     *
338     *    construction du champs de commande.
339     *
340              LDAA    xmi_sv
341              ANDA    %P
342              ORAA    %X'OF
343     *
344     *    mise du champs de commande dans le buffer d'emission.
345     *
346              STAA    ()
347     *
348     *    preparation du compteur DMA et saut a la routine
349     *    d'emission d'une trame.
350     *
351              LDAA    %X'02
352              STAA    BCROL
353              CLR     BCROH
354              JMP     EMIDMA
355     *
356     *    on va envoyer une trame DISC.
357     *    on va a la routine de construction de l'adresse.
358     *
359     TRADSC  JSR     ADRR
360     *
```

```
361     *   constuction du champs de commande.
362     *
363             LDAA    xmi_sv
364             ANDA    %P
365             ORAA    %X'43
366     *
367     *   mise du champs de commande dans le buffer d'emission.
368     *
369             STAA    ()
370     *
371     *   preparation du compteur DMA et saut a la routine
372     *   d'emission d'une trame.
373     *
374             LDAA    %X'02
375             STAA    BCROL
376             CLR     BCROH
377             JMP     EMIDMA
378     *
379     *   on va envoyer une trame UA.
380     *   on va a la routine de construction de l'adresse.
381     *
382     TRAUA   JSR     ADRL
383     *
384     *   construction du champs de commande.
385     *
386             LDAA    xmi_sv
387             ANDA    %F
388             ORAA    %X'63
389     *
390     *   mise du champs de commande dans le buffer d'emission.
391     *
392             STAA    ()
393     *
394     *   preparation du compteur DMA et saut a la routine
395     *   d'emission d'une trame.
396     *
397             LDAA    %X'02
398             STAA    BCROL
399             CLR     BCROH
400             JMP     EMIDMA
401     *
402     *   on va envoyer une trame CMDR.
403     *   on va a la routine de construction de l'adresse.
404     *
405     TRACDR  JSR     ADRL
406     *
407     *   construction du champs de commande.
408     *
409             LDAA    xmi_sv
410             ANDA    %F
411             ORAA    %X'87
412     *
413     *   mise du champs de commande dans le buffer d'emission.
414     *
415             STAA    ()
416     *
417     *   mise des 3 bytes d'information a 0 dans le buffer
418     *   d'emission.
419     *
420             INX
```

```
421             CLR     ()
422             INX
423             CLR     ()
424             INX
425             CLR     ()
426     *
427     *   preparation du compteur DMA et saut a la routine
428     *   d'emission d'une trame.
429     *
430             LDAA    %X'05
431             STAA    BCROL
432             CLR     BCROH
433             JMP     EMIDMA
434     *
435     *   routine de construction de l'adresse AD_LOC dans
436     *   le THR.
437     *
438     ADRL    LDAA    %AD_LOC
439             STAA    THR
440             RTS
441     *
442     *   routine de construction de l'adresse AD_REM dans
443     *   le THR.
444     *
445     ADRR    LDAA    %AD_REM
446             STAA    THR
447             RTS
448     *
449     *   routine d'emission d'une trame.
450     *   mise de l'HDLC en mode DATA.
451     *
452     EMIDMA  LDAA    %X'C1
453             COMA
454             STAA    CR1
455     *
456     *   preparation du registre adresse du DMA.
457     *
458             LDX     %xmi_bu
459             STX     AROH
460     *
461     *   on relance le DMA.
462     *
463             LDAA    PC
464             ORAA    %X'01
465             STAA    PC
466             RTS
467     *
468     *   on avait I dans xmi_rq : on va essayer d'
469     *   envoyer une trame d'information.
470     *
471     ESSAI   JSR     ESSTRI
472     *
473     *   la variable TSTESS nous signale si on a
474     *   reussi a envoyer une trame d'information.
475     *
476             LDAA    TSTESS
477     *
478     *   si on a pu envoyer une trame d'information,
479     *   on sort immediatement de la routine, sinon on
480     *   envoie un flag avant de sortir.
```

```
481     *
482             BNE     SORTIE
483     *
484             LDAA    %FLAG
485             STAA    xmi_sv
486             LDAA    %X'E1
487             COMA
488             STAA    CR1
489     *
490     SORTIE  RTS
491     *
492     *   on est dans la routine d'essai d'envoi d'
493     *   une trame d'information.
494     *   la variable xmi_cd nous renseigne sur le cas
495     *   a traiter.
496     *
497     ESSTRI  LDAA    xmi_cd
498             BNE     C1
499             JMP     CMD0
500     C1      CMPA    %X'01
501             BNE     C2
502             JMP     CMD1
503     C2      CMPA    %X'02
504             BNE     C3
505             JMP     CMD2
506     C3      CMPA    %X'03
507             BNE     ERROR6
508             JMP     CMD3
509     *
510     *   la variable xmi_cd a pris une valeur
511     *   anormale : on le signale en mettant 6 dans
512     *   ZERROR.
513     *
514     ERROR6  LDAA    %X'06
515             STAA    ZERROR
516             JMP     X'F800
517     *
518     *   on veut envoyer une trame d'information avec
519     *   xmi_cd = 0. L'emission n'est donc pas autorisee
520     *   et on met TSTESS a 0 pour signaler que l'
521     *   emission n'a pas eu lieu.
522     *
523     CMD0    CLR     TSTESS
524             RTS
525     *
526     *   on veut envoyer une trame d'information avec
527     *   xmi_cd = 1. On est donc dans le cas d'une
528     *   emission normale en sequence.
529     *   On commence par tester pour voir s'il reste des
530     *   paquets non encore emis dans le buffer. S'il n'en
531     *   reste pas, on sort en signalant qu'on  n'a pas
532     *   pu emettre.
533     *   le test est le suivant :
534     *   (bo_pct > ((v_send - v_nack + 8 ) & 07))
535     *
536     CMD1    LDAB    v_send
537             SUBB    v_nack
538             ADDB    %X'08
539             ANDB    %X'07
540             LDAA    bo_pct+1
```

```
541             CBA
542             BHI     BON
543             JMP     RATE
544     *
545     *   il reste des paquets a emettre dans le buffer.
546     *   on va regarder si on ne depasse pas l'
547     *   anticipation. Si oui, on sort en signalant qu'
548     *   on n'a pas pu emettre.
549     *   le test est le suivant :
550     *   (v_send != ((v_nack + K) & 07))
551     *
552     BON     LDAA    v_send
553             LDAB    v_nack
554             ADDB    %K
555             ANDB    %X'07
556             CBA
557             BNE     CHARGE
558             JMP     RATE
559     *
560     *   si la longueur du paquet est nulle
561     *   on met a jour le compteur de places libres
562     *   bo_fct et on remet le pointeur bu_out au
563     *   debut du buffer. Sinon, on va directement en
564     *   CALFEN.
565     *
566     CHARGE  LDX     bo_opt
567             LDAA    ()
568             BNE     CALFEN
569     *
570     *   on fait le calcul suivant :
571     *   bo_fct =+ (bu_out + BOSIZE - bo_opt)
572     *
573             LDAB    %bu_out
574             LDAA    %bu_out/256
575             ADDB    %BOSIZE
576             ADCA    %BOSIZE/256
577             SUBB    bo_opt+1
578             SBCA    bo_opt
579             ADDB    bo_fct+1
580             ADCA    bo_fct
581             STAB    bo_fct+1
582             STAA    bo_fct
583     *
584             LDX     %bu_out
585             STX     bo_opt
586     *
587     *   on met a jour la fenetre en faisant :
588     *   window [v_send] = bo_opt
589     *
590     CALFEN  LDAA    v_send
591             ASLA
592             STAA    ZTRAV+1
593             CLR     ZTRAV
594             LDX     ZTRAV
595     *
596             LDAA    bo_opt
597             LDAB    bo_opt+1
598             STAA    window()
599             STAB    window+1()
600     *
```

```
601     *   on va a la routine de construction de l'
602     *   adresse.
603     *
604             LDX     %xmi_bu
605             JSR     ADRR
606     *
607     *   on construit le champs de commande et on
608     *   le met dans le buffer.
609     *
610             LDAA    v_rec
611             ASLA
612             ASLA
613             ASLA
614             ASLA
615             ORAA    v_send
616             ASLA
617             STAA    ()
618             INX
619             STX     bx_ipt
620     *
621     *   on charge dans a le compteur pour la
622     *   boucle de transfert.
623     *
624             LDX     bo_opt
625             LDAA    ()
626             INX
627             STX     bo_opt
628     *
629     *   on garni le compteur du DMA
630     *
631             STAA    BCROL
632             INC     BCROL
633             INC     BCROL
634             CLR     BCROH
635     *
636     *   on effectue la boucle du transfert du
637     *   paquet de bu_out dans xmi_bu le buffer d'
638     *   emission
639     *
640     BOUCL1  LDX     bo_opt
641             LDAB    ()
642             INX
643             STX     bo_opt
644             LDX     bx_ipt
645             STAB    ()
646             INX
647             STX     bx_ipt
648             DECA
649             BNE     BOUCL1
650     *
651     *   on met I dans xmi_sv
652     *
653             LDAA    %I
654             STAA    xmi_sv
655     *
656     *   on signale qu'on a reussi l'emission en mettant
657     *   la variable TSTESS a 1 et on va a la routine
658     *   d'emission
659     *
660             CLR     TSTESS
```

```
661             INC     TSTESS
662             JSR     EMIDMA
663             RTS
664     *
665     *   on n'a pu effectuer l'emission : on met
666     *   la variable TSTESS a 0
667     *
668     RATE    CLR     TSTESS
669             RTS
670     *
671     *   on essaie d'envoyer une trame d'information
672     *   et xmi_cd = 2.
673     *   on va relancer l'emission a partir de v_nack
674     *   on met xmi_cd a 1 et on fait v_send = v_nack
675     *
676     CMDB    CLR     xmi_cd
677             INC     xmi_cd
678             LDAA    v_nack
679             STAA    v_send
680     *
681     *   on met a jour bo_opt en faisant :
682     *   bo_opt = window [v_send]
683     *
684             LDAB    v_send
685             ASLB
686     *
687             STAB    ZTRAV+1
688             CLR     ZTRAV
689             LDX     ZTRAV
690             LDAA    window+1()
691             LDAB    window()
692             STAA    bo_opt+1
693             STAB    bo_opt
694     *
695     *   on va a la routine de construction de l'
696     *   adresse
697     *
698             LDX     %xmi_bu
699             JSR     ADRR
700     *
701     *   on construit le champs de commande
702     *
703             LDAA    v_rec
704             ASLA
705             ASLA
706             ASLA
707             ASLA
708             ORAA    v_send
709             ASLA
710             STAA    ()
711             INX
712             STX     bx_ipt
713     *
714     *   on charge dans A le compteur pour la
715     *   boucle de transfert
716     *
717             LDX     bo_opt
718             LDAA    ()
719             INX
720             STX     bo_opt
```

```
721     *
722     *   on met a jour le compteur du DMA
723     *
724             STAA    BCROL
725             INC     BCROL
726             INC     BCROL
727             CLR     BCROH
728     *
729     *   on transfert le paquet de bu_out dans
730     *   xmi_bu le buffer d'emission
731     *
732     BOUCL2  LDX     bo_opt
733             LDAB    0,X
734             INX
735             STX     bo_opt
736             LDX     bx_ipt
737             STAB    0,X
738             INX
739             STX     bx_ipt
740             DECA
741             BNE     BOUCL2
742     *
743     *   on met I dans xmi_sv
744     *
745             LDAA    %I
746             STAA    xmi_sv
747     *
748     *   on signale qu'on a reussi l'emission
749     *   en mettant TSTESS a la valeur 1
750     *   et on va a la routine d'emission
751     *
752             CLR     TSTESS
753             INC     TSTESS
754             JSR     EMIDMA
755             RTS
756     *
757     *   on essaie d'envoyer une trame d'information
758     *   et xmi_cd = 3. On est dans le cas d'une reprise
759     *   sur temporisateur.
760     *   on commence par mettre xmi_cd a 0
761     CMD3    CLR     xmi_cd
762     *
763     *   on met a jour bo_tpt en faisant :
764     *   bo_tpt = window [v_nack]
765     *
766             LDAB    v_nack
767             ASLB
768     *
769             STAB    ZTRAV+1
770             CLR     ZTRAV
771             LDX     ZTRAV
772             LDAA    window+1,X
773             LDAB    window,X
774             STAA    bo_tpt+1
775             STAB    bo_tpt
776     *
777     *   on va a la routine de construction de l'
778     *   adresse
779     *
780             LDX     %xmi_bu
```

```
781                 JSR     ADRR
782     *
783     *   on construit dans le buffer le champs de
784     *   commande
785     *
786                 LDAA    v_rec
787                 ASLA
788                 ASLA
789                 ASLA
790                 ASLA
791                 ORAA    v_send
792                 ASLA
793                 ORAA    %X'10
794                 STAA    ()
795                 INX
796                 STX     bx_ipt
797     *
798     *   on met dans le registre A le compteur de
799     *   transfert
800     *
801                 LDX     bo_tpt
802                 LDAA    ()
803                 INX
804                 STX     bo_tpt
805     *
806     *   on met a jour le compteur du DMA
807     *
808                 STAA    BCROL
809                 INC     BCROL
810                 INC     BCROL
811                 CLR     BCROH
812     *
813     *   on effectue la boucle du transfert
814     *   du paquet de bu_out dans le buffer d'
815     *   emission xmi_bu
816     *
817     BOUCL3      LDX     bo_tpt
818                 LDAB    ()
819                 INX
820                 STX     bo_tpt
821                 LDX     bx_ipt
822                 STAB    ()
823                 INX
824                 STX     bx_ipt
825                 DECA
826                 BNE     BOUCL3
827     *
828     *   on met I_BIS dans xmi_sv
829     *
830                 LDAA    %I_BIS
831                 STAA    xmi_sv
832     *
833     *   on met TSTESS a 1 pour signaler qu'on a
834     *   reussi l'emission et on branche a la routine
835     *   d'emission
836     *
837                 CLR     TSTESS
838                 INC     TSTESS
839                 JSR     EMIDMA
840                 RTS
```

```
841     *
842     ************************************************
843     *                                              *
844     *   routine de fin d'emission avec erreur      *
845     *                                              *
846     ************************************************
847     *
848     *   on ne devrait normalement jamais aboutir a
849     *   cette routine
850     *   si on y arrive cela signifie une erreur
851     *   hardware : le THR n'est pas rempli assez
852     *   rapidement par le HDLC
853     *   on met 3 dans ZERROR et on retourne au
854     *   moniteur
855     XMWIUN  LDAA    %X'03
856             STAA    ZERROR
857             JMP     X'F800
858     ****************************************
859     *                                      *
860     *   routine de fin de reception DMA    *
861     *                                      *
862     ****************************************
863     *
864     *   arriver dans cette routine constitue une
865     *   erreur car cela signifie que la trame est
866     *   trop longue.
867     *   en effet, le compteur DMA en reception est
868     *   initialise de maniere telle que pour une
869     *   trame normale on n'ait jamais de fin de
870     *   reception DMA
871     *   on met 7 dans ZERROR et on retourne au
872     *   moniteur
873     *
874     CASCR1  LDAA    %X'07
875             STAA    ZERROR
876             JMP     X'F800
877     ****************************************
878     *                                      *
879     *   routine de fin d'emission DMA      *
880     *                                      *
881     ****************************************
882     *
883     *   arriver dans cette routine signifie qu'
884     *   on a terminer le transfert des donnees
885     *   d'une trame.
886     *   il suffit donc de mettre le HDLC en mode
887     *   FCS afin de terminer la trame.
888     *
889     CASCR0  LDAA    %X'F0
890             COMA
891             STAA    CR1
892             RTS
```

Aug 25, 1980 13:00

Apat

```
    1     #         seg       L
    2     intNR     zmb       1
    3     trbuO     zmb       140
    4     trbu1     zmb       140
    5     n_rec     zmb       1
    6     cmdes     zmb       1
    7     bu_in     zmb       1536
    8     v_rec     zmb       1
    9     state     zmb       1
   10     lastNR    zmb       1
   11     bi_fct    zmb       2
   12     rec_lg    zmb       1
   13     bo_fct    zmb       2
   14     adr_er    zmb       1
   15     maj_ct    zmb       2
   16     xmi_ab    zmb       1
   17     bi_pct    zmb       2
   18     v_nack    zmb       1
   19     stateS    zmb       1
   20     stateP    zmb       1
   21     xmi_cd    zmb       1
q  22     bi_ipt    zmb       2
   23     bo_pct    zmb       2
   24     n_send    zmb       1
   25     clo_ct    zmb       2
   26     bi_opt    zmb       2
   27     bo_ipt    zmb       2
   28     v_send    zmb       1
   29     tri_fl    zmb       1
   30     rep_ct    zmb       1
   31     bo_opt    zmb       2
   32     xmi_fl    zmb       1
   33     tro_fl    zmb       1
   34     mes_rq    zmb       1
   35     drr_ct    zmb       1
   36     drr_fl    zmb       1
   37     drs_ct    zmb       1
   38     drs_fl    zmb       1
   39     pdp_ct    zmb       2
   40     xmi_bu    zmb       140
   41     bu_out    zmb       1536
   42     trf_pt    zmb       2
   43     xmi_rq    zmb       1
   44     drr_st    zmb       1
   45     tri_pt    zmb       2
   46     xmi_ct    zmb       1
   47     bx_ipt    zmb       2
   48     bx_opt    zmb       2
   49     xmi_sv    zmb       1
   50     put_ct    zmb       1
   51     tro_pt    zmb       2
   52     window    zmb       16
   53     rec_bu    zmb       140
   54     rec_ct    zmb       1
   55     br_ipt    zmb       2
   56     br_opt    zmb       2
   57     dlr_st    zmb       1
   58     timout    zmb       1
   59     *
   60     TDR       equ       X'DFC1
```

```
 61     RDR       equ     X'DFC1
 62     CONTRG    equ     X'DFC0
 63     STATRG    equ     X'DFC0
 64     CRAB      equ     X'4003
 65     CRBB      equ     X'4007
 66     CRAH      equ     X'4002
 67     CRBH      equ     X'4006
 68     ORAB      equ     X'4001
 69     ORBB      equ     X'4005
 70     ORAH      equ     X'4000
 71     ORBH      equ     X'4004
 72     SR        equ     X'5802
 73     NOSIGA    equ     60
 74     NOSIGB    equ     60
 75     AD_LOC    equ     3
 76     ENTETE    equ     128
 77     BISIZE    equ     1536
 78     AD_REM    equ     1
 79     BOSIZE    equ     1536
 80     F         equ     16
 81     I         equ     9
 82     K         equ     7
 83     P         equ     16
 84     TRSIZE    equ     140
 85     VIDANGE   equ     2
 86     MESSAGE   equ     0
 87     N2        equ     10
 88     TRANSFE   equ     1
 89     T1        equ     500
 90     UA        equ     7
 91     RR        equ     1
 92     REJ       equ     3
 93     RNR       equ     2
 94     RUN       equ     0
 95     FLAG      equ     0
 96     DISC      equ     6
 97     SIGA      equ     52
 98     SIGB      equ     52
 99     CMDR      equ     8
100     SARM      equ     5
101     PMIN      equ     3
102     PMAX      equ     131
103     SLEEP     equ     2
104     STAND     equ     1
105     I_BIS     equ     10
106               org     X'1000
```

Aug 25, 1980 13:02

Aparcd.mas

```
  1     #         seg       L
  2               ORG       X'0005
  3     intNR     zmb       1
  4     trbu0     zmb       140
  5     trbu1     zmb       140
  6     n_rec     zmb       1
  7     cmdes     zmb       1
  8     bu_in     zmb       1536
  9     v_rec     zmb       1
 10     state     zmb       1
 11     lastNR    zmb       1
 12     bi_fct    zmb       2
 13     rec_lg    zmb       1
 14     bo_fct    zmb       2
 15     adr_er    zmb       1
 16     maj_ct    zmb       2
 17     xmi_ab    zmb       1
 18     bi_pct    zmb       2
 19     v_nack    zmb       1
 20     stateS    zmb       1
 21     stateP    zmb       1
 22     xmi_cd    zmb       1
 23     bi_ipt    zmb       2
 24     bo_pct    zmb       2
 25     n_send    zmb       1
 26     clo_ct    zmb       2
 27     bi_opt    zmb       2
 28     bo_ipt    zmb       2
 29     v_send    zmb       1
 30     tri_fl    zmb       1
 31     rep_ct    zmb       1
 32     bo_opt    zmb       2
 33     xmi_fl    zmb       1
 34     tro_fl    zmb       1
 35     mes_rq    zmb       1
 36     drr_ct    zmb       1
 37     drr_fl    zmb       1
 38     drs_ct    zmb       1
 39     drs_fl    zmb       1
 40     pdp_ct    zmb       2
 41     xmi_bu    zmb       140
 42     bu_out    zmb       1536
 43     trf_pt    zmb       2
 44     xmi_rq    zmb       1
 45     drr_st    zmb       1
 46     tri_pt    zmb       2
 47     xmi_ct    zmb       1
 48     bx_ipt    zmb       2
 49     bx_opt    zmb       2
 50     xmi_sv    zmb       1
 51     put_ct    zmb       1
 52     tro_pt    zmb       2
 53     window    zmb       16
 54     rec_bu    zmb       140
 55     rec_ct    zmb       1
 56     br_ipt    zmb       2
 57     br_opt    zmb       2
 58     dlr_st    zmb       1
 59     timout    zmb       1
 60     bo_tpt    ZMB       2
```

```
   61     *                                                              *
   62     TDR       equ     X'DFC1
   63     RDR       equ     X'DFC1
   64     CONTRG    equ     X'DFC0
   65     STATRG    equ     X'DFC0
   66     NOSIGA    equ     60
   67     NOSIGB    equ     60
   68     AD_LOC    equ     1
   69     ENTETE    equ     128
   70     BISIZE    equ     1536
   71     AD_REM    equ     3
   72     BOSIZE    equ     1536
   73     F         equ     16
   74     I         equ     9
   75     K         equ     7
   76     P         equ     16
   77     TRSIZE    equ     140
   78     VIDANGE   equ     2
   79     MESSAGE   equ     0
   80     N2        equ     10
   81     TRANSFE   equ     1
   82     T1        equ     500
   83     UA        equ     7
   84     RR        equ     1
   85     REJ       equ     3
   86     RNR       equ     2
   87     RUN       equ     0
   88     FLAG      equ     0
   89     DISC      equ     6
   90     SIGA      equ     52
   91     SIGB      equ     52
   92     CMDR      equ     8
   93     SARM      equ     5
   94     PMIN      equ     3
6  95     PMAX      equ     131
   96     SLEEP     equ     2
   97     STAND     equ     1
   98     I_BIS     equ     10
   99     ZERROR    ZMB     1
  100     TSTESS    ZMB     1
  101     TRB       ZMB     1
  102     ZTRAV     ZMB     2
  103     ZONTRA    ZMB     1
  104     *
  105     *
  106     *
  107     CR1       EQU     X'5807
  108     CR2       EQU     X'5806
  109     CR3       EQU     X'5805
  110     SR        EQU     X'5802
  111     IR        EQU     X'5803
  112     THR       EQU     X'5803
  113     RHR       EQU     X'5804
  114     AR        EQU     X'5804
  115     *
  116     *                                                              *
  117     *
  118     ORAH      EQU     X'4000
  119     DDRAH     EQU     X'4000
  120     ORAB      EQU     X'4001
```

```
121     DDRAB   EQU     X'4001
122     CRAH    EQU     X'4002
123     CRAB    EQU     X'4003
124     DDRBH   EQU     X'4004
125     ORBH    EQU     X'4004
126     DDRBB   EQU     X'4005
127     ORBB    EQU     X'4005
128     CRBH    EQU     X'4006
129     CRBB    EQU     X'4007
130     *
131     *
132     *
133     ORA     EQU     X'5000
134     DDRA    EQU     X'5000
135     CRA     EQU     X'5002
136     *
137     *
138     *
139     AR0H    EQU     X'DFE0
140     AR0L    EQU     X'DFE1
141     BCR0H   EQU     X'DFE2
142     BCR0L   EQU     X'DFE3
143     AR1H    EQU     X'DFE4
144     AR1L    EQU     X'DFE5
145     BCR1H   EQU     X'DFE6
146     BCR1L   EQU     X'DFE7
147     AR2H    EQU     X'DFE8
148     AR2L    EQU     X'DFE9
149     BCR2H   EQU     X'DFEA
150     BCR2L   EQU     X'DFEB
151     AR3H    EQU     X'DFEC
152     AR3L    EQU     X'DFED
153     BCR3H   EQU     X'DFEE
154     BCR3L   EQU     X'DFEF
155     CCR0    EQU     X'DFF0
156     CCR1    EQU     X'DFF1
157     CCR2    EQU     X'DFF2
158     CCR3    EQU     X'DFF3
159     PC      EQU     X'DFF4
160     IC      EQU     X'DFF5
161     DCR     EQU     X'DFF6
162             org     X'1000
```

Aug 25, 1980 13:04

Apartd.mas

```
    1     #         seg       L
    2     *****     ADRESSES POUR ETTD
    3               ORG       X'0005
    4     intNR     zmb       1
    5     trbu0     zmb       140
    6     trbu1     zmb       140
    7     n_rec     zmb       1
    8     cmdes     zmb       1
    9     bu_in     zmb       1536
   10     v_rec     zmb       1
   11     state     zmb       1
   12     lastNR    zmb       1
   13     bi_fct    zmb       2
   14     rec_lg    zmb       1
   15     bo_fct    zmb       2
   16     adr_er    zmb       1
   17     maj_ct    zmb       2
   18     xmi_ab    zmb       1
   19     bi_pct    zmb       2
   20     v_nack    zmb       1
   21     stateS    zmb       1
   22     stateP    zmb       1
   23     xmi_cd    zmb       1
   24     bi_ipt    zmb       2
   25     bo_pct    zmb       2
   26     n_send    zmb       1
   27     clo_ct    zmb       2
I  28     bi_opt    zmb       2
   29     bo_ipt    zmb       2
   30     v_send    zmb       1
   31     tri_fl    zmb       1
   32     rep_ct    zmb       1
   33     bo_opt    zmb       2
   34     xmi_fl    zmb       1
   35     tro_fl    zmb       1
   36     mes_rq    zmb       1
   37     drr_ct    zmb       1
   38     drr_fl    zmb       1
   39     drs_ct    zmb       1
   40     drs_fl    zmb       1
   41     pdp_ct    zmb       2
   42     xmi_bu    zmb       140
   43     bu_out    zmb       1536
   44     trf_pt    zmb       2
   45     xmi_rq    zmb       1
   46     drr_st    zmb       1
   47     tri_pt    zmb       2
   48     xmi_ct    zmb       1
   49     bx_ipt    zmb       2
   50     bx_opt    zmb       2
   51     xmi_sv    zmb       1
   52     put_ct    zmb       1
   53     tro_pt    zmb       2
   54     window    zmb       16
   55     rec_bu    zmb       140
   56     rec_ct    zmb       1
   57     br_ipt    zmb       2
   58     br_opt    zmb       2
   59     dlr_st    zmb       1
   60     timout    zmb       1
```

```
  61     bo_tpt  ZMB     2
  62     *                                                              *
  63     TDR     equ     X'DFC1
  64     RDR     equ     X'DFC1
  65     CONTRG  equ     X'DFC0
  66     STATRG  equ     X'DFC0
  67     NOSIGA  equ     60
  68     NOSIGB  equ     60
  69     AD_LOC  equ     3
  70     ENTETE  equ     128
  71     BISIZE  equ     1536
  72     AD_REM  equ     1
  73     BOSIZE  equ     1536
  74     F       equ     16
  75     I       equ     9
  76     K       equ     7
  77     P       equ     16
  78     TRSIZE  equ     140
  79     VIDANGE equ     2
  80     MESSAGE equ     0
  81     N2      equ     10
  82     TRANSFE equ     1
  83     T1      equ     500
  84     UA      equ     7
  85     RR      equ     1
  86     REJ     equ     3
  87     RNR     equ     2
  88     RUN     equ     0
  89     FLAG    equ     0
  90     DISC    equ     6
  91     SIGA    equ     52
  92     SIGB    equ     52
  93     CMDR    equ     8
  94     SARM    equ     5
  95     PMIN    equ     3
  96     PMAX    equ     131
  97     SLEEP   equ     2
  98     STAND   equ     1
  99     I_BIS   equ     10
 100     ZERROR  ZMB     1
 101     TSTESS  ZMB     1
 102     TRB     ZMB     1
 103     ZTRAV   ZMB     2
 104     ZONTRA  ZMB     1
 105     *
 106     *
 107     *
 108     CR1     EQU     X'5807
 109     CR2     EQU     X'5806
 110     CR3     EQU     X'5805
 111     SR      EQU     X'5802
 112     IR      EQU     X'5803
 113     THR     EQU     X'5803
 114     RHR     EQU     X'5804
 115     AR      EQU     X'5804
 116     *
 117     *
 118     *
 119     DRAH    EQU     X'4000
 120     DDRAH   EQU     X'4000
```

```
121     ORAB    EQU     X'4001
122     DDRAB   EQU     X'4001
123     CRAH    EQU     X'4002
124     CRAB    EQU     X'4003
125     DDRBH   EQU     X'4004
126     ORBH    EQU     X'4004
127     DDRBB   EQU     X'4005
128     ORBB    EQU     X'4005
129     CRBH    EQU     X'4006
130     CRBB    EQU     X'4007
131     *
132     *
133     *
134     ORA     EQU     X'5000
135     DDRA    EQU     X'5000
136     CRA     EQU     X'5002
137     *
138     *
139     *
140     AR0H    EQU     X'DFE0
141     AR0L    EQU     X'DFE1
142     BCR0H   EQU     X'DFE2
143     BCR0L   EQU     X'DFE3
144     AR1H    EQU     X'DFE4
145     AR1L    EQU     X'DFE5
146     BCR1H   EQU     X'DFE6
147     BCR1L   EQU     X'DFE7
148     AR2H    EQU     X'DFE8
149     AR2L    EQU     X'DFE9
150     BCR2H   EQU     X'DFEA
151     BCR2L   EQU     X'DFEB
152     AR3H    EQU     X'DFEC
153     AR3L    EQU     X'DFED
154     BCR3H   EQU     X'DFEE
155     BCR3L   EQU     X'DFEF
156     CCR0    EQU     X'DFF0
157     CCR1    EQU     X'DFF1
158     CCR2    EQU     X'DFF2
159     CCR3    EQU     X'DFF3
160     PC      EQU     X'DFF4
161     IC      EQU     X'DFF5
162     DCR     EQU     X'DFF6
163             org     X'1000
```

```
    1                   seg     L
    2       init68
    3                   des
    4                   des
    5                   tsx
    6                   ldab    %bu_out
    7                   ldaa    %bu_out/256
    8                   stab    bo_ipt+1
    9                   staa    bo_ipt
   10                   ldab    %bu_out
   11                   ldaa    %bu_out/256
   12                   stab    bo_opt+1
   13                   staa    bo_opt
   14                   ldab    %0
   15                   ldaa    %6
   16                   stab    bo_fct+1
   17                   staa    bo_fct
   18                   clr     bo_pct
   19                   clr     bo_pct+1
   20                   ldab    %bu_in
   21                   ldaa    %bu_in/256
   22                   stab    bi_ipt+1
   23                   staa    bi_ipt
   24                   ldab    %bu_in
   25                   ldaa    %bu_in/256
   26                   stab    bi_opt+1
   27                   staa    bi_opt
   28                   ldab    %0
   29                   ldaa    %6
   30                   stab    bi_fct+1
   31                   staa    bi_fct
   32                   clr     bi_pct
   33                   clr     bi_pct+1
   34                   clr     xmi_cd
   35                   clr     state
   36                   clr     rec_ct
   37                   ldab    %trbu0
   38                   ldaa    %trbu0/256
   39                   stab    tri_pt+1
   40                   staa    tri_pt
   41                   ldx     tri_pt
   42                   clr     0()
   43                   clr     tri_fl
   44                   ldab    %rec_bu
   45                   ldaa    %rec_bu/256
   46                   stab    br_ipt+1
   47                   staa    br_ipt
   48                   clr     dlr_st
   49                   clr     drs_ct
   50                   ldab    %3
   51                   stab    CONTRG
   52       *                                                          *
   53                   clr     CRAH
   54                   clr     CRAB
   55                   clr     CRBH
   56                   clr     CRBB
   57                   clr     ORAH
   58                   clr     ORAB
   59                   ldab    %255
   60                   stab    ORBH
```

```
61                  ldab    %255
62                  stab    ORBB
63                  ldab    %60
64                  stab    CRAB
65                  ldab    %60
66                  stab    CRBH
67                  ldab    %169
68                  stab    CONTRO
69                  ldab    %55
70                  stab    CRAH
71                  ldab    %55
72                  stab    CRBB
73      L1
74                  ins
75                  ins
76                  rts
```

initcd

```
    1                   seg     L
    2       init68
    3                   des
    4                   des
    5                   tsx
    6                   ldab    %bu_out
    7                   ldaa    %bu_out/256
    8                   stab    bo_ipt+1
    9                   staa    bo_ipt
   10                   ldab    %bu_out
   11                   ldaa    %bu_out/256
   12                   stab    bo_opt+1
   13                   staa    bo_opt
   14                   ldab    %0
   15                   ldaa    %6
   16                   stab    bo_fct+1
   17                   staa    bo_fct
   18                   clr     bo_pct
   19                   clr     bo_pct+1
   20                   ldab    %bu_in
   21                   ldaa    %bu_in/256
   22                   stab    bi_ipt+1
   23                   staa    bi_ipt
   24                   ldab    %bu_in
   25                   ldaa    %bu_in/256
   26                   stab    bi_opt+1
   27                   staa    bi_opt
   28                   ldab    %0
   29                   ldaa    %6
   30                   stab    bi_fct+1
   31                   staa    bi_fct
   32                   clr     bi_pct
   33                   clr     bi_pct+1
   34                   clr     xmi_cd
   35                   clr     state
   36                   clr     rec_ct
   37                   ldab    %trbu0
   38                   ldaa    %trbu0/256
   39                   stab    tri_pt+1
   40                   staa    tri_pt
   41                   ldx     tri_pt
   42                   clr     0,x
   43                   clr     tri_fl
   44                   ldab    %rec_bu
   45                   ldaa    %rec_bu/256
   46                   stab    br_ipt+1
   47                   staa    br_ipt
   48                   clr     dlr_st
   49                   clr     drs_ct
   50                   clr     CRAH
   51                   clr     CRAB
   52                   clr     CRBH
   53                   clr     CRBB
   54                   clr     DRAH
   55                   clr     DRAB
   56       *
   57                   ldab    %255
   58                   stab    DRBH
   59                   ldab    %255
   60                   stab    DRBB
```

```
                ldab    %60
                stab    CRAB
                ldab    %60
                stab    CRBH
                ldab    %55
                stab    CRAH
                ldab    %55
                stab    CRBB
*
*
*
                LDAB    %X'04
                STAB    CRA
*
                LDAA    %X'22
                STAA    ORA
*
                CLR     CRA
*
                LDAA    %X'27
                STAA    DDRA
*
                STAB    CRA
*
*
*
REP105          LDAA    SR
                COMA
                BITA    %X'40
                BEQ     REP105
*
                LDAA    %X'01
                COMA
                STAA    CR2
*
                LDAA    %X'E0
                COMA
                STAA    CR1
*
                LDX     %X'0040
DELAI           DEX
                NOP
                BNE     DELAI
*
                LDAA    %X'F0
                STAA    ORA
*
*
*
                LDAA    %X'08
                STAA    DCR
*
                LDAA    %X'04
                STAA    CCR1
*
                INCA
                STAA    CCR0
*
                LDX     %trbu0
                INX
```

initcd

```
121                 STX     AR1H
122     *
123                 LDAA    %145
124                 STAA    BCR1L
125                 CLR     BCR1H
126     *
127                 CLR     TRB
128     *
129                 LDAA    PC
130                 ORAA    %X'02
131                 STAA    PC
132     *
133                 LDAA    IC
134                 ORAA    %X'03
135                 STAA    IC
136     *
137                 CLR     ZERROR
138     L1
139                 ins
140                 ins
141                 rts
```

```
    1               seg     L
    2   init68
    3               des
    4               des
    5               tsx
    6               ldab    %bu_out
    7               ldaa    %bu_out/256
    8               stab    bo_ipt+1
    9               staa    bo_ipt
   10               ldab    %bu_out
   11               ldaa    %bu_out/256
   12               stab    bo_opt+1
   13               staa    bo_opt
   14               ldab    %0
   15               ldaa    %6
   16               stab    bo_fct+1
   17               staa    bo_fct
   18               clr     bo_pct
   19               clr     bo_pct+1
   20               ldab    %bu_in
   21               ldaa    %bu_in/256
   22               stab    bi_ipt+1
   23               staa    bi_ipt
   24               ldab    %bu_in
   25               ldaa    %bu_in/256
   26               stab    bi_opt+1
   27               staa    bi_opt
   28               ldab    %0
   29               ldaa    %6
   30               stab    bi_fct+1
   31               staa    bi_fct
   32               clr     bi_pct
   33               clr     bi_pct+1
   34               clr     xmi_cd
   35               clr     state
5  36               clr     rec_ct
   37               ldab    %trbu0
   38               ldaa    %trbu0/256
   39               stab    tri_pt+1
   40               staa    tri_pt
   41               ldx     tri_pt
   42               clr     0()
   43               clr     tri_fl
   44               ldab    %rec_bu
   45               ldaa    %rec_bu/256
   46               stab    br_ipt+1
   47               staa    br_ipt
   48               clr     dlr_st
   49               clr     drs_ct
   50               clr     CRAH
   51               clr     CRAB
   52               clr     CRBH
   53               clr     CRBB
   54               clr     ORAH
   55               clr     ORAB
   56   *
   57               ldab    %255
   58               stab    ORBH
   59               ldab    %255
   60               stab    ORBB
```

```
  61                ldab    %60
  62                stab    CRAB
  63                ldab    %60
  64                stab    CRBH
  65                ldab    %55
  66                stab    CRAH
  67                ldab    %55
  68                stab    CRBB
  69    *
  70    *
  71    *
  72    *
  73    *
  74    *         Initialisation SDLC
  75    *
  76                LDAA    %X'01
  77                COMA
  78                STAA    CR1
  79    *
  80    REP107      LDAA    SR
  81                COMA
  82                BITA    %X'20
  83                BEQ     REP107
  84    *
  85                LDAA    IR
  86    *
  87                LDAA    %X'01
  88                COMA
  89                STAA    CR2
  90    *
  91                LDAA    %X'E1
  92                COMA
  93                STAA    CR1
  94    *
  95    REP106      LDAA    IR
  96                COMA
  97                BITA    %X'02
  98                BEQ     REP106
  99    *
 100    *
 101    *
 102    *
 103    *
 104                LDAA    %X'08
 105                STAA    DCR
 106    *
 107                LDAA    %X'04
 108                STAA    CCR1
 109    *
 110                INCA
 111                STAA    CCR0
 112    *
 113                LDX     %trbu0
 114                INX
 115                STX     AR1H
 116    *
 117                LDAA    %145
 118                STAA    BCR1L
 119                CLR     BCR1H
 120    *
```

```
121                 CLR     TRB
122     *
123                 LDAA    PC
124                 ORAA    %X'02
125                 STAA    PC
126     *
127                 LDAA    IC
128                 ORAA    %X'03
129                 STAA    IC
130     *
131                 CLR     ZERROR
132     L1
133                 ins
134                 ins
135                 rts
```

Aug 25, 1980 13:05

Atrame

```
    1              seg     L
    2    start     lds     %x'D3FF
    3              jsr     init68
    4              cli
    5              jsr     automa
    6              ldaa    %15
    7              staa    lastNR
    8              swi
    9    stop      tsx
   10              ldaa    3(
   11              staa    lastNR
   12              swi
   13    waitin    wai
   14              rts
   15    interu    ldaa    CONTRG
   16              anda    %1
   17              beq     L1
   18              jbs     vidrec
   19    L1        ldaa    CONTRG
   20              anda    %2
   21              beq     L2
   22              jbs     vidsen
   23    L2        ldaa    CRAH
   24              bge     L3
   25              jbs     paqrec
   26              ldaa    CRBB
   27              bge     interu
   28              jbs     paqsen
   29              bra     interu
   30    L3        ldaa    CRBB
   31              bge     L4
   32              jbs     paqsen
   33              bra     interu
   34    L4        cli
   35              rti
   36    startT    ldaa    %X'37
   37              staa    CRAH
   38              ldaa    %X'01
   39              staa    timout
   40              ldaa    %X'3F
   41              staa    CRAH
   42              rts
   43    stopT     ldaa    %X'37
   44              staa    CRAH
   45              clr     timout
   46              rts
   47    ignore    rts
```

```
   1                seg     L
   2     start      lds     %x'D3FF
   3                jsr     init68
   4                cli
   5                jsr     automa
   6                ldaa    %15
   7                staa    lastNR
   8                swi
   9     stop       tsx
  10                ldaa    3()
  11                staa    lastNR
  12                swi
  13     waitin     wai
  14                rts
  15     INTERU     LDAA    IR
  16                COMA
  17                BITA    %1
  18                BEQ     TSTDMA
  19     INTRQ1     PSHA
  20                BITA    %X'80
  21                BEQ     SUITE1
  22                JSR     RENDER
  23                BRA     SUITE2
  24     SUITE1     BITA    %X'40

 25                 BEQ     SUITE2
  26                JSR     REWIER
  27     SUITE2     PULA
  28                BITA    %X'20
  29                BEQ     SUITE3
  30                JSR     XMNOER
  31                BRA     TSTDMA
  32     SUITE3     BITA    %X'10
  33                BEQ     TSTDMA
  34                JSR     XMWIUN
  35     TSTDMA     LDAA    IC
  36                BITA    %X'80
  37                BEQ     TSTPIA
  38     TSTCR1     LDAA    CCR1
  39                BITA    %X'80
  40                BEQ     TSTCRO
  41                JSR     CASCR1
  42     TSCCRO     LDAA    CCRO
  43                BITA    %X'80
  44                BEQ     TSTPIA
  45                JSR     CASCRO
  46                BRA     TSTPIA
  47     TSTCRO     LDAA    CCRO
  48                BITA    %X'80
  49                BEQ     ERRDR1
  50                JSR     CASCRO
  51     *
  52     TSTPIA     ldaa    CRAH
  53                bge     L3
  54                jbs     paqrec
  55                ldaa    CRBB
  56                bge     INTERU
  57                jbs     paqsen
  58                bra     INTERU
  59     L3         ldaa    CRBB
  60                bge     L4
```

```
61                  jbs     paqsen
62                  bra     INTERU
63      L4          cli
64                  rti
65      startT      ldaa    %X'37
66                  staa    CRAH
67                  ldaa    %X'01
68                  staa    timout
69                  ldaa    %X'3F
70                  staa    CRAH
71                  rts
72      stopT       ldaa    %X'37
73                  staa    CRAH
74                  clr     timout
75                  rts
76      ignore      rts
77      xmista      RTS
78      ERROR1      LDAA    %X'01
79                  STAA    ZERROR
80                  JMP     X'F800
```